Tabu Search for Bus and Train Driver Scheduling With Time Windows

by

Yindong Shen

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy

The University of Leeds
School of Computing

May 2001

The candidate confirms that the work submitted is her own and that appropriate credit has been given where reference has been made to the work of others.

Acknowledgements

I would like to thank my supervisors, Dr. Raymond S. K. Kwan and Professor Anthony Wren, for their help, guidance and encouragement. I am also grateful to Dr. Ann S. K. Kwan, for her assistance and providing me with the test problems. I would like to thank Dr. Angseng Li for his advice on the mathematical presentations. I would also like to thank Dr. Sarah Fores for her comments on and suggestions to the thesis.

I would like to thank my mother Mrs. Siying Shen; she loves me deeply and is always ready to provide me with help.

Finally, my thanks go to my husband Mr. Daniel Jiahong Xia, for his support and patience. I am most grateful to my lovely son, Ben, who is nine years old. His accompany has provided me enormous comfort, happiness and support.

I dedicate this thesis to Ben.

Abstract

The bus and train driver scheduling problem involves assigning bus or train work to drivers in such a way that all the bus or train work is covered and the number of drivers and duty costs are minimised. This is complicated by the fact that there are many restrictions on the duty generation.

The generate-and-select approach is at present the most successful for bus and train driver scheduling. It involves generating a set of legal potential driver duties from which a minimal and most efficient subset is selected. Filtering rules are often applied so that the set of potential duties generated would not be prohibitively large. Moreover, *windows of relief opportunities* (WROs), which provide ranges of opportunities for relieving drivers, are beyond the capability of being handled by the existing systems. The usual practice is to consider one, sometimes two, discrete times within each time window. Optimality of solution is therefore compromised.

The research presented in this thesis focuses on solving the driver scheduling problem with WROs using a constructive approach, which builds and refines a single schedule iteratively. Filtering rules are unnecessary under the approach. The 2-opt heuristic approach is first investigated, during which the potential of constructive heuristics is explored. Based on the experience, the Tabu Search meta-heuristic approach is then investigated. Multi-neighbourhoods and an appropriate memory scheme, which are essential elements of Tabu Search are designed and tailored for the driver scheduling problem with WROs. Alternative designs have been tested and compared with best known solutions drawn from real-life data sets. The tabu search approach is very fast, can handle WROs, and has achieved results comparable to those based on mathematical programming approaches. Taking advantage of WROs, it can improve best known solutions obtained by the existing systems. Consequently, it could be incorporated into existing systems to improve the solution by taking advantage of WROs.

Declarations

Some parts of the work presented in this thesis have been published or will appear in the following articles:

Shen, Y. and Kwan, R.S.K. (2001). Tabu Search for Driver Scheduling. To appear in: Voß, S. and Daduna, J.R. (eds.) *Computer-Aided Scheduling of Public Transport*, Springer Verlag.

Shen, Y. and Kwan, R.S.K. (2001). A constructive approach to bus and train driver scheduling. In: *Proceedings of the IIE Annual Conference 2001*, May 20-23, 2001, Dallas, Texas.

Shen, Y. and Kwan, R.S.K. (2000). A Tabu Search algorithm with 2-opt moves for bus and rail driver scheduling. Presented in: *the Eleventh Young Operational Research Conference*, 28th-30th March, 2001, Fitzwilliam College, Cambridge.

Shen, Y. and Kwan, R.S.K. (2000). Tabu Search for time windowed public transport driver scheduling. *Technical Report 2000.14*, School of Computing, University of Leeds.

Contents

Acknowledgements **i**

Abstract **ii**

Declarations **iii**

Contents **iv**

List of Tables **xi**

List of Figures **xii**

1 Introduction **1**

1.1 Bus and train driver scheduling problems 1

1.1.1 Vehicle work 2

1.1.2 Vehicle work with windows of relief opportunities 4

1.1.3 Driver scheduling 5

1.1.4 Duty types 6

1.1.5 Labour agreement rules 7

1.1.6 Objectives of driver scheduling 10

1.2 Computerised driver scheduling approaches 10

1.3 Purpose of the research 11

1.4 Organisation of the thesis 12

2 Review of Bus and Train Driver Scheduling Approaches 15

2.1 Introduction 15

2.2 Early heuristic approaches 16

2.2.1 Elias' scheduling systems 17

2.2.2 Weaver and Wren's work 19

2.2.3 The TRACS system 21

2.2.4 RUCUS and RUCUS II 23

2.2.5 The COMPACS system 25

2.2.6 Overview of the early heuristic approaches 25

2.3 Mathematical programming approaches 26

2.3.1 Mathematical model of set covering and set partitioning 26

2.3.2 The IMPACS system 28

2.3.3 The TRACS II system 30

2.3.4 The Crew-opt method of the HASTUS system 33

2.3.5 Overview of the mathematical programming approaches 35

2.4 Meta-heuristic approaches 36

2.4.1 A Tabu Search approach - Cavique et al 36

2.4.2 Genetic Algorithms 38

2.4.2.1 Kwan et al 38

2.4.2.2 Li and Kwan 39

2.4.3 A Simulated Evolution algorithm – Li and Kwan 40

2.4.4 An Ant System - Forsyth and Wren 41

2.4.5 Overview of the meta-heuristic approaches 42

2.5 Constraint programming approaches 42

2.5.1 Layfield et al 42

2.5.2 Curtis et al 43
2.5.3 Overview of the constraint programming approaches 44
2.6 Recent transport scheduling packages 44
2.7 Conclusions – generate-and-select approach vs. constructive approach 46

3 Neighbourhood Search Heuristics for Driver Scheduling 49

3.1 Introduction 49
3.2 Neighbourhood Search 51
3.3 2-opt heuristics 53
3.3.1 Vehicle scheduling versus driver scheduling 54
3.3.2 A bus vehicle scheduling system – BOOST 55
3.3.3 2-opt heuristics for driver scheduling 55
3.4 Simplified driver scheduling problem 59
3.5 Outline of the 2-opt driver scheduling approach 63
3.5.1 Modelling driver activities and duty 63
3.5.2 General scheme of the 2-opt driver scheduling approach 64
3.6 Formation of an initial schedule 65
3.7 Swapping links 68
3.7.1 2-opt swapping links 68
3.7.2 Link swapping rules 70
3.7.2.1 Swapping for improving the current schedule 71
3.7.2.2 Swapping for creating favourable conditions for improvement . 71
3.7.3 Experiments on 2-opt swapping 73
3.7.4 K-opt swapping links 74
3.7.5 Summary 77

3.8 Replacing AROs 77
3.8.1 Principle of replacing AROs 77
3.8.2 Strategies of replacing AROs 79
3.8.3 Hybrid of swapping links and replacing AROs 82
3.9 Adding duties 83
3.10 Implementation and results 85
3.10.1 Greater Manchester Buses 85
3.10.2 Merseyside Transport 86
3.10.3 Regional Railways North East 87
3.10.4 Summary 88
3.11 Conclusions 88

4 Modelling the Driver Scheduling Problem and Overview of HACS 91

4.1 Introduction 91
4.2 A formal model for driver scheduling 92
4.2.1 Vehicle work 92
4.2.2 A driver schedule 93
4.2.3 Labour agreement rules and costing functions 96
4.2.3.1 Duty types 97
4.2.3.2 Costing a duty 98
4.2.3.3 Costing a link 101
4.2.3.4 Costing a schedule 101
4.2.4 Objectives 102
4.3 An alternative model of a driver schedule 103
4.4 Spell-model versus piece-model 105

4.5 Overview of the HACS approach 106
4.5.1 Construction of an initial schedule 107
4.5.2 Refinement of the schedule 107
4.6 Conclusions 108

5 Tabu Search Technique Tailored for the Driver Scheduling Problem with WROs 109

5.1 Introduction 109
5.2 Tabu Search 111
5.3 Objective functions 112
5.4 Multi-neighbourhoods based on the spell-model 113
5.4.1 Swapping two links 113
5.4.2 Swapping two spells 115
5.4.3 Inserting one spell 116
5.4.4 Recutting block 117
5.4.4.1 Definition of the potential alternatives of a given ARO 117
5.4.4.2 Definition of the predecessor and successor of an ARO without considering WROs 118
5.4.4.3 Conversion of a time window into discrete times 119
5.4.4.4 Enhanced definitions of the predecessor and successor of an ARO 120
5.4.4.5 Definition of recutting-block operation 121
5.4.5 Summary 123
5.5 Multi-neighbourhoods with combined utilisation of the spell model and the piece-model 123

5.6 Memory schemes 125
5.6.1 Move representation and move attribute 126
5.6.2 Move value 126
5.6.3 Tabu tenure and tabu status 127
5.6.4 Candidate list and tabu list 128
5.6.4.1 Elite candidate list 129
5.6.4.2 An efficient memory scheme for driver scheduling 132
5.7 Tabu Search algorithms in HACS 134
5.8 Conclusions 138

6 Evaluation and Computational Experiments 139

6.1 Introduction 139
6.2 Test problems and the best known solutions 140
6.3 HACS based on spell-model 143
6.3.1 Experiments on different neighbourhood structures 143
6.3.2 Experiments on a steepest descent method 145
6.3.3 Evaluation of the swapping processes 146
6.3.4 Summary 150
6.4 HACS with combined utilisation of the spell-model and the piece-model 151
6.5 Demonstration of handling WROs 152
6.5.1 The TEST problem and the HACS solution 153
6.5.2 TRACS II solutions to the TEST problem and analyses 154
6.5.3 Summary 158
6.6 Experiments on real-life problems with WROs 158
6.7 Experiments to improve best known solutions 160
6.8 Conclusions 162

7 Conclusions **164**

7.1 Summary 164

7.2 Research Contributions 166

7.3 Future Work 168

Bibliography **170**

Glossary **181**

List of Tables

1.1 Example of PNB rules in train operation 9

3.1 Neighbourhood Search Method 51
3.2 Descent Method 52
3.3 Steepest Descent Method 52
3.4 Comparison between 2-opt and TRACS II solutions to the GMB problem 86
3.5 Comparison between 2-opt and TRACS II solutions to the GMBmini problem 86
3.6 Comparison between 2-opt and TRACS II solutions to the MTLD problem 86
3.7 Comparison between 2-opt and TRACS II solutions to the MTLDmini problem 87
3.8 Comparison between 2-opt and TRACS II solutions to the RRNE problem 88

5.1 Tabu Search 112

6.1 Properties and best known solutions of test problems 141
6.2 Results obtained by HACS with different single swapping neighbourhood structure..... 144
6.3 The solutions obtained by HACS with multi-neighbourhoods 144
6.4 Results obtained by a Steepest Descent method 145
6.5 The starting solutions and the corresponding final solutions to D6 obtained by HACS based on the spells in the TRACS II solutions 148
6.6 The starting solutions and the corresponding final solutions to D7 obtained by HACS based on the spells in the TRACS II solutions 149
6.7 The starting solutions and the corresponding final solutions to D9 obtained by HACS based on the spells in the TRACS II solutions 150
6.8 Results obtained by the HACS approach with the spell-based and piece-based operations combined 152
6.9 HACS solutions involving WROs 159
6.10 HACS solutions based on the best known solutions 160

List of Figures

1.1 Example of a block in a vehicle schedule 3
1.2 Example of a vehicle block with relief opportunities 4
1.3 Example of a vehicle block with windows of relief opportunities 5
1.4 Illustration of a driver schedule 6

2.1 Workbench main window of TRACS II 30
2.2 Different levels of RO reduction 33
2.3 Different levels of potential duty reduction 33

3.1 Hierarchy of driver activities 64
3.2 Illustration of a duty with two spells 64
3.3 The major functional components of the 2-opt heuristics 66
3.4 Illustration of 2-opt swapping-link (standard) 69
3.5 Illustration of 2-opt swapping-link (a) 69
3.6 Illustration of 2-opt swapping-link (b) 70
3.7 Illustration of 2-opt swapping-link (c) 70
3.8 Illustration of the possible effect of the WideBR swap 72
3.9 Illustration of 3-opt swapping-link 75
3.10 Illustration of 4-opt swapping-link 75
3.11 Illustration of 5-opt swapping-link 76
3.12 Illustration of replacing AROs 78
3.13 Types of invalid duties and the relevant AROs for replacement 80
3.14 Illustration of replacing AROs on three links 82
3.15 Example of two-link-adding-duty 84

4.1 Illustration of a driver duty 94
4.2 Relationship between blocks and duties 96
4.3 Classification of driver duties 98

5.1 Illustration of swapping two links 114
5.2 Example of turning a spell into an end spell 114
5.3 Illustration of swapping two spells 116
5.4 Illustration of inserting one spell 117
5.5 Duty with a selected link and its WROs 120
5.6 Illustration of recutting blocks 122
5.7 Illustration of three invalid links involved in *move*(*i*,*j*) and *move*(*i*,*x*) 130
5.8 Illustration of the new links after two elite candidate moves 130

6.1 The TEST problem and the HACS solution 153
6.2 Blocks in the TEST problem after shrinking each WRO into a RO 154
6.3 TRACS II solution to the TEST problem after shrinking each WRO into a RO 155
6.4 TRACS II solution to the TEST problem after expanding each time window into individual minutes 157
6.5 Illustration of a vehicle work with WROs in the London Underground Metropolitan Line 161

Chapter One

Introduction

1.1 Bus and train driver scheduling problems

The bus and train driver scheduling problem involves finding the most efficient way of assigning notional drivers to the daily operations of a fleet of vehicles. Bus driver scheduling and train driver scheduling are different aspects of the same problem (Kwan et al., 2000). All the features that are in the bus operation can be found in the train operation but not vice versa (Kwan, 1999). In both cases, driver duties are made up of spells of work on one or more vehicles, and all the vehicle work must be covered by such duties. The term *vehicle* shall be used to denote a bus or a train throughout this thesis.

The precursor to the driver scheduling problem is the vehicle scheduling problem, in which vehicles are allocated to the journeys to be operated (the service is fixed in advance, and will remain unchanged for usually months at a time). Since both the vehicle scheduling problem and

the driver scheduling problem are individually hard, the usual practice is to compile schedules separately for the vehicles and for the drivers. When all daily driver duties for a schedule or set of schedules (sometimes weekday and weekend have different schedules) have been complied it is necessary to combine them into sets of work for actual drivers on a weekly basis. In the United Kingdom, the weekly duties are usually organised by the management into a rotating roster, and the drivers work through all the weekly rotas in turn.

The driver scheduling problem presented here is concerned with the assignment of notional drivers to a predetermined bus or train schedule. The number of possible combinations for partitioning the vehicle work in a schedule is usually astronomical. There is a set of restrictions on the driver schedules. For example, daily working time for a driver has to be limited to a certain number of hours; a driver is normally required to have a break if the continuous driving time has reached a certain limit. These restrictions are called *labour agreement rules*, which vary a great deal between different operators. The most common rules are listed in Section 1.1.5. These rules governing the legality of a driver duty affect profoundly the complexity of duty compilation. The driver schedule must be legal according to the rules. It should use the minimum number of duties and/or have the lowest total cost.

1.1.1 Vehicle work

In a vehicle schedule, the work is usually presented as a set of *blocks*. *A block* presents a sequence of journeys to be operated by one vehicle during one day, beginning with a *pull-out* from, and ending with a *pull-in*, to a depot. The journeys of a vehicle that leaves and returns to the depot more than once in a day make up several blocks. Figure 1.1 displays an example of a block in a vehicle schedule, where *layover* denotes the time allowance during which a vehicle waits at a stop after its arrival and before starting its next journey; *empty run* denotes

unproductive vehicle work, during which a vehicle runs from one location to another without carrying passengers.

Block 1:

Start from Depot at GNFDRL 05:31

Location	Arrival Time	Departure Time	Layover (minutes)
GNFDRL		05:31	
BULBTS	06:03	06:07	:4
GNFDRL	06:46	06:59	:13
BULBTS	07:33	07:37	:4
GNFDRL	08:23	08:26	:3
HVKEHR	08:54	08:55	:1
GNFDRL	09:33	09:42	:9
HVKEHR	10:10	10:11	:1
GNFDRL	10:46	10:56	:10
BULBTS	11:32	11:37	:5
GNFDRL	12:21	12:27	:6
HVKEHR	12:55	12:56	:1
GNFDRL	13:31	13:36	:5
BULBTS	14:12	14:17	:5
GNFDRL	15:01	15:12	:11
HVKEHR	15:40	15:41	:1
GNFDRL	16:21	16:27	:6
HVKEHR	16:55	16:56	:1
GNFDRL	17:36	17:36	:0
BULBTS	18:12	18:17	:5
GNFDRL	18:59	19:16	:17
BULBTS	19:48	19:53	:5
GNFDRL	20:32	20:35	:3
HVKEHR	20:54	20:55	:1
GNFDRL	21:20	21:35	:15
HVKEHR	21:54	21:55	:1
GNFDRL	22:20	22:35	:15
HVKEHR	22:54	22:55	:1
GNFDRL	23:20	23:20	empty run :4
SMFMHR	23:24	23:25	:1
EALCKR	23:55	24:00	:5
SMFMHR	24:32	24:32	empty run :4
GNFDRL	24:36		

Arrival back Depot at GNFDRL 24:36

Figure 1.1: Example of a block in a vehicle schedule

For driver scheduling purposes, a block is usually represented in a graphical format (see Parker and Smith, 1981) showing the vehicle number (i.e. block number) and all the vehicle work (including empty runs) in the block while identifying a sequence of *relief opportunities* (RO for short). A *RO* is a time/location pair, at which drivers can be relieved. The time is called a *relief time*, while the location is called a *relief point* and coded by a single alphabet in a block graph.

Not all the places where a vehicle stops in a vehicle schedule are feasible places for relieving drivers. Only some of the places are designated as relief points by transport operators. Figure 1.2 shows an example, which represents the block in Figure 1.1 in a graphical format identifying relief opportunities. In this example, we suppose that only GNFDRL and BULBTS are designated as relief points, which are denoted as G and B respectively in Figure 1.2.

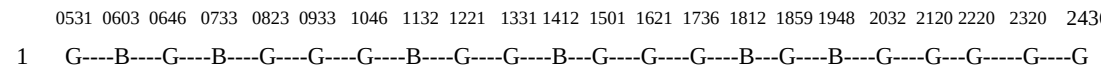


Figure 1.2: Example of a vehicle block with relief opportunities

In Figure 1.2, only the arrival times at relief points (plus the starting time at the depot) are selected as relief times. This is at present the most common practice in bus and train driver scheduling. However, a more comprehensive representation that takes into account the gaps between the arrivals and the departures at relief points is provided in the next section.

1.1.2 Vehicle work with windows of relief opportunities

In practice, when a vehicle stops at a location, there is often a gap between the arrival time and the departure time, which is called a *time window*. In bus vehicle scheduling, the time window is called a *layover* (see Figure 1.1). A time window at a particular relief point constitutes a *window of relief opportunities* (WRO for short), during which drivers can be relieved at any time within the window. Some WROs are called *attended WROs* (A-WRO for short), during which the vehicle must be attended by one driver. The others (normally long time gaps) are called *unattended WROs* (U-WRO for short), during which the vehicle can be left without a driver. For train operations, extra time has to be allowed for the driver to power off and on the engine before and after an U-WRO. Figure 1.3 exemplifies a vehicle block with WROs, where

7909 is the block number; the thick line denotes vehicle work while the rectangles denote WROs; above these are their time ranges while below are the codes of their relief points. A-WROs and U-WROs are distinguished by whether the thick line passes through their corresponding rectangles. A *RO*, denoted by a small circle, is a special case of a WRO, in which the arrival time is the same as the departure time. This special case of WROs will be distinguished and indicated in this thesis only when necessary. Any time in a time window together with the relief point constitutes a time-point pair and is also called a *RO*. When WROs are considered, a RO denotes a time-point pair in a WRO. The work between two consecutive WROs on a block is defined as a *piece of work*, which is not permitted to be broken up.

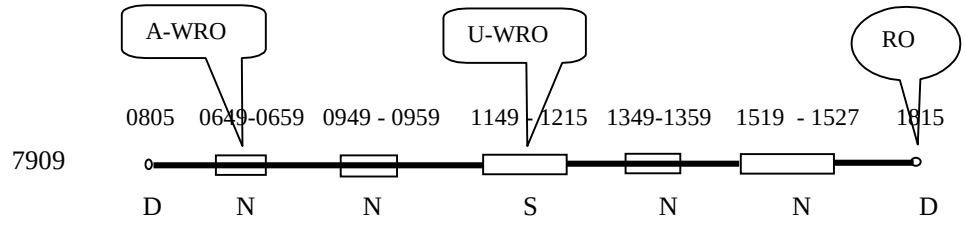


Figure 1.3: Example of a vehicle block with windows of relief opportunities

1.1.3 Driver scheduling

Driver scheduling is a process of partitioning vehicle work into a set of legal *driver duties*, the *schedule*, such that the total number of duties and/or the total duty costs are minimised.

A *duty* is the work to be performed by a driver during one day from signing on until signing off at a depot. Each duty consists of a sequence of driver activities, i.e. a *sign-on* activity, a set of *spells*, and a *sign-off* activity. A *spell* is a section of continuous vehicle work to be operated by a driver without a break. Each *spell* includes at least two ROs, the first and the last ROs are called *active relief opportunities* (*ARO*s for short), during which the drivers are actually relieved. Figure 1.4 illustrates a driver schedule, where '+' denotes an ARO, while G and H are

relief point codes. Driver scheduling without considering windows of relief opportunities can be regarded as a special case.

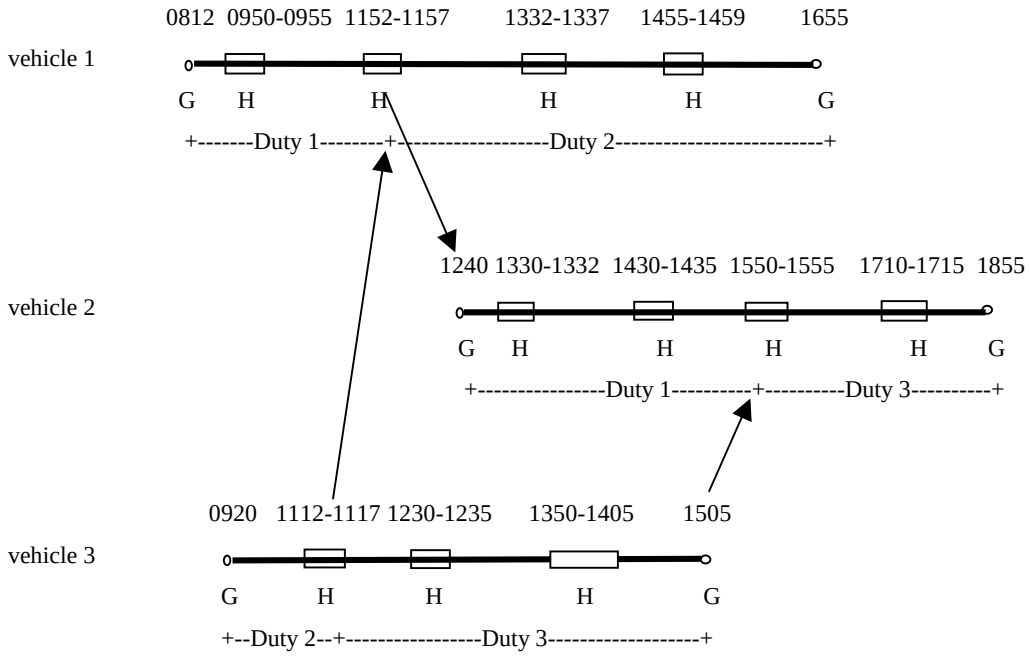


Figure 1.4: Illustration of a driver schedule

1.1.4 Duty types

Duties can be classified into two principal types: *split duties* and *straight duties*. A *split duty* is one with a longer spreadover than standard duties, e.g. up to 12 hours or more, plus a long break in the middle. The other duties are called *straight duties*, which may be further divided into different types according to which time period the duty covers or to which range of spreadover length the duty falls in. For instance, in bus operations, *straight duties* can be divided into *early duties* (taking early buses out of the depot and covering part of the morning peak), *late duties* (taking late buses into the depot and covering part of the afternoon peak), and *middle duties*. In UK organisations, straight duties usually contain two stretches separated by a

meal break. There may also be a few straight duties containing two or more meal breaks, and some, which are usually only two to five hours long containing a straight run without mealbreak. In North America, straight bus runs without mealbreak are common.

The above is a general classification; other classifications exist. In practice, the classification for a particular problem is defined by the operator and each type of duty often has a set of labour agreement rules to govern its legality. A duty satisfying all the governing rules is called a *valid duty*, a *legal duty* or a *feasible duty*; otherwise it is called an *invalid duty*, an *illegal duty* or an *infeasible duty*.

1.1.5 Labour agreement rules

A feasible schedule must be legal according to labour agreement rules determined by the government, the transport operators and the union. Some other local unwritten rules may also have to be enforced. Labour agreement rules vary considerably between operators. The common rules for bus drivers are listed in (Smith and Wren, 1988) and are included in the following list for both bus and train drivers.

1) A driver starts work by signing on and finishes work by signing off at the same depot.

2) There is a limit on *spreadover*, i.e. the length of a duty, which is the length of time from first signing on to finally signing off, and usually no more than twelve hours. Each kind of duty has a maximum length of spreadover, while some have a limit on the minimum length of spreadover.

3) There is a maximum *spell*, i.e. length of driving time that a driver can work continuously without a break on one vehicle. There may be periods within a spell when the vehicle stops and attendance by a driver is optional. Some operations stipulate two different rules to restrict the maximum length of a spell with or without a stop. The maximum spell with a stop is always equal or longer than those without a stop.

4) There is a maximum *stretch*, i.e. length of time that a driver can work without a meal break, say four to five hours. A stretch (e.g., from signing on to the start of the first meal break, or from the end of the last meal break to signing off, or between two successive meal breaks) consists of one or more successive spells in a duty. Between two successive spells in a stretch there is an intervening short break, called a *join-up*. This break must be long enough for the driver to travel to take over the following spell of work.

5) *Meal break* (i.e. the time allowance between two spells of work for the driver to travel to the canteen, have a meal and then take over the next vehicle) must be a minimum length of time. In bus operations, there is usually one meal break in a duty of say at least 30 or 40 minutes. For a split duty, the longest gap between two consecutive spells must not be shorter than the due minimum length of time. In UK train operation, a meal break is usually called a *PNB* (*Physical Needs Break*) and there may be one or more PNBs in a duty. The rules governing when PNBs should occur and how long they should at least be are complicated. Table 1.1 shows the PNB rules in a train operation, where PNB-PERIODS denotes the time ranges when PNBs can occur, while LENGTH denotes the minimum length of a PNB. Both PNB-PERIODS and LENGTH depend on the spreadover of a duty. For instance, the first row in Table 1.1 means: one PNB of at least 30 minutes must occur between the third and fifth hour relative to the start of the duty if

the spreadover of the duty is equal to or less than eight hours. The last row means: two PNBs of 20 minutes each must occur between the second and the seventh hour if the spreadover of the duty is greater than eight hours and less than or equal to nine hours.

Table 1.1: Example of PNB rules in train operation

No. PNBS	LENGTH	Spreadover-RANGE		PNB-PERIODS	
1	:30	000	800	300	500
1	:30	801	900	200	600
2	:20	000	800	200	600
2	:20	801	900	200	700

6) There is a maximum *working time* for a duty. The calculation of working time of a duty depends on the operator and the type of the duty. For some duties, the time from signing on to signing off is fully counted as working time; for some duties, non-driving time, e.g. mealbreak and the signing on and off allowances before and after a long meal break, is not counted as working time; for some duties, part of non-driving time is counted as working time.

7) There are usually restrictions on the time of earliest sign-on/sign-off, earliest start on/finish off, latest sign-on/sign-off, and latest start on/finish off dependent on the duty types.

The above rules govern the legality of a duty and must be adhered to in a feasible driver schedule. In addition to the labour agreement rules, some operators may have other requirements governing the legality of a duty. For instance, a driver must have knowledge of the type of vehicle he/she drives. We shall use the term *labour agreement rules* to denote any rules and requirements governing the legality of a duty throughout this thesis.

1.1.6 Objectives of driver scheduling

Although objectives of driver scheduling may vary, forming a feasible schedule covering all the vehicle work is the most essential objective for all operators and must be fulfilled. (In practice, operators may accommodate some infeasibility by bending the rules or adjusting the vehicle trips when considerable saving can be made). Minimising the number of duties required and the total duty cost are primary objectives for most operators. These two objectives sometimes conflict and a trade-off may be needed. In the UK, minimising the total number of duties usually has priority over minimising total duty cost.

1.2 Computerised driver scheduling approaches

Early computerised driver scheduling approaches were purely heuristic. They were heavily reliant on domain knowledge, and often needed large amounts of manual intervention such that the algorithms were not readily portable to different transport operators. By the 1980's it was generally recognised that heuristics alone were not suitable for general use (Wren and Rousseau, 1995). The purely heuristic approach was therefore abandoned in favour of mathematical programming approaches aided by heuristics. The principle mathematical programming approach, e.g. TRACS II (Fores and Proll, 1998; Kwan et al., 1999 and 2000; Fores et al., 2001), first generates all the 'good' valid potential driver duties according to labour agreement rules, then if necessary reduces the very large potential duty set using heuristics, and lastly uses a set covering or set partitioning formulation to select the cheapest subset of duties that covers each piece of vehicle work. This approach can be labelled as a *generate-and-select* approach. The mathematically based generate-and-select approach has been successful in many practical systems (e.g. Rousseau and Desrochers, 1995; Fores et al., 1999 and 2001), and is still dominant in commercial application of driving scheduling. The advances in the mathematical

programming solvers, such as column generation techniques (Rousseau and Desrochers, 1995; Fores, 1996; Fores et al., 1998 and 1999), have made the mathematically-based approaches more powerful in solving large problems. The column generation technique can reduce the burden of duty generation if further potential duties are generated as required during the selection process (Rousseau and Desrochers, 1995).

Despite their successes, the current systems can neither be regarded as black boxes that produce working schedules completely automatically nor guarantee to yield optimal solutions. As one of the most successful systems, TRACS II (Fores et al., 1999; Kwan et al., 2000) has, through long development working with many operators, reached a level of generality such that it can fit with many companies' requirements. Filtering rules are often applied as parameters so that the set of potential duties generated would not be prohibitively large. These parameters have to be manipulated to compile driver schedules for different transport operators. Such manipulation can be frustrating to schedulers who do not appreciate the practical limitations of the mathematical programming approach. Moreover, *windows of relief opportunities* are beyond the capability of the current systems. The reason is that WROs greatly enlarge the sizes of problems. The existing systems only handle relief opportunities as discrete times, and it would be impractical to expand explicitly each time window into individual minutes. The usual practice is to consider one, or sometimes a few, discrete times within each time window. Optimality of the schedule is therefore compromised.

1.3 Purpose of the research

The bus and train driver scheduling problem is of great practical importance as efficient schedules can make huge monetary savings. Building efficient schedules is one of the best ways for the transportation industry to maximise profit. This problem has attracted much research

interest since the 1960's (Elias, 1964; Wren, 1968; Wilson, 1999; Voß and Daduna, 2001). Although a great deal of progress has been made, the problem (known as a hard combinatorial optimisation problem) still has scope for new research and improvements.

This research aims to investigate a *constructive heuristic approach* that can tackle the driver scheduling problem with windows of relief opportunities. The proposed approach aims:

1) To handle windows of relief opportunities;
2) To produce solutions not requiring the use of any artificial rules as necessitated in the generate-and-select approach.

A *constructive heuristic approach* builds and refines a single schedule. We use the term "constructive" to contrast with the term *generate-and-select*. The term "constructive approach" is used throughout this thesis. Under the class of the constructive approaches, artificial rules are unnecessary. Obviously for the constructive approach, the fewer rules that are imposed on the validity of a driver duty, the easier it is to construct a solution. Moreover, the more relief opportunities that are provided in the blocks, the more chance it has to refine the schedule. Windows of relief opportunities provide ranges of opportunities for drivers to change over. The constructive approach could benefit from the windows of relief opportunities to refine schedules with more flexibility in recutting blocks.

1.4 Organisation of the thesis

Following this introduction, Chapter two reviews a number of computerised approaches to the driver scheduling problem. The other chapters, except the Conclusions, present the research in

stages by investigating and building a suitable approach to solve the driver scheduling problem with WROs.

Chapter three presents some initial research on a 2-opt heuristic approach for driver scheduling. The driver scheduling problem was simplified, in which up to two spells were allowed in a duty and the WROs were reduced to one or two ROs. The main aims of this initial research were to investigate the potential of constructive heuristics, which were anticipated to be able to handle WROs for driver scheduling and to obtain some experience in building the proposed approach of this research.

Chapter four shows two models: *spell-model* and *piece-model*, which aim to represent the real-life driver scheduling problem with WROs. *Spell-model* is first built by a natural presentation of the problem, in which a spell is the basic unit of driving work in a duty. During the scheduling process, the compositions of spells are variable. For the sake of developing the new algorithm, a *piece-model* is also built, in which fixed *pieces of work* are the basic units. It is anticipated that a combined utilisation of the two models would be needed. Based on the models, a driver scheduling approach called HACS is developed, an overview of which is presented.

Chapter five describes in detail the Tabu Search technique tailored for the driver scheduling problem with WROs, constituting the core of the HACS approach. Tabu Search is a general scheme that must be tailored to the details of the problem at hand. This tailoring process is non-trivial. Strategic use of memory and design of neighbourhood structures are critical. The driver scheduling problem without WROs is a special case of that with WROs.

Chapter six presents a series of experiments and computational results compared with best known solutions drawn from real-life data sets. Some components of HACS are evaluated while the ability of handling WROs is assessed by experiments.

Finally, Chapter seven sums up the contributions of the research and considers options for future research.

Chapter Two

Review of Bus and Train Driver Scheduling Approaches

2.1 Introduction

Scheduling public transport drivers using computers began in the 1960's. Since then a great deal of progress has been made and many approaches have been developed. Wren and Rousseau (1995) gave an overview of the approaches, many of which have been reported in a series of proceedings of the eight international conferences on Computer-Aided Scheduling of Public Transport (edited by Bodin and Bergman, 1975; Wren, 1981; Rousseau, 1985; Daduna and Wren, 1988; Desrochers and Rousseau, 1992; Daduna, et al., 1995; Wilson, 1999; Voß and Daduna, 2001). Research into bus and train driver scheduling approaches has been aided by the continuing rapid growth in computer technology. The approaches may be roughly divided into four groups: *earlier heuristics*, *mathematical programming approaches*, *meta-heuristics* and *constraint programming approaches*. The classification is not distinctive because mathematical

programming approaches often involve heuristics while *meta-heuristics* often incorporate some simple heuristics. Constraint programming approaches model practical problems as constraint satisfaction problems, which are solved by heuristics or mathematical programming techniques.

Some earlier heuristic systems are described in Section 2.2 while some successful systems based on mathematical programming approaches are in Section 2.3. Sections 2.4 and 2.5 describe some recent work on *meta-heuristics* and *constraint programming*. Section 2.6 briefly reports three recent transport scheduling packages exhibited in the latest international conference on Computer-Aided Scheduling of Public Transport. A concluding comparison between the generate-and-select approach and the constructive approach is given at the end of this chapter.

2.2 Early heuristic approaches

The earliest attempts (up till the 1970s) to solve driver scheduling problems used heuristics. The reasons were that the computer was not powerful enough for running the mathematical solvers, and the techniques employed by the mathematical programming solvers were not as advanced as nowadays. Most heuristic systems at that time constructed a schedule using methods similar to those used by manual schedulers. The refinement of the schedule was made either by the schedulers interactively or by automatic heuristic procedures making limited alterations.

This section reviews the early attempts made by Elias and by Weaver and Wren, the more sophisticated TRACS and RUCUS systems, and an interactive system called COMPACS. The review of other heuristic systems in this period can be found in the theses of Smith (1986),

Willers (1995), Fores (1996), Kwan (1999) and Curtis (2000)). These early heuristic approaches are summarised at the end of this section.

2.2.1 Elias' scheduling systems

Elias applied the computer to aid manual schedulers to compile bus driver schedules in the early 1960's (Elias, 1964). The situation that Elias' aid system dealt with was common in the United States, in which many duties were straight runs, i.e. a single spell of bus work of about 8 hours in length. The remaining work was combined into two-bus duties with an intervening break, and some might be left as 'trippers', which were single spells of work to be covered as overtime.

Elias' aid system includes three phases:

1) *Form straight runs*

   A straight run is first cut from the beginning of the blocks, the duration of which is longer than 8 hours. If there are no relief opportunities giving a straight run exactly 8 hours long, the nearest previous or next relief opportunity is selected depending on the cost of the duty. The remaining work of the block is left as a spell if it is less than 8 hours; otherwise, another straight run is cut from the end of the block. Once every block is cut, the above process will be repeated, by cutting the straight runs from the end of each block. Lastly, a subset of the straight runs is selected by human schedulers.

2) *Form 2-bus duties*

   Combine the remaining spells into a set of all possible two-spell duties, from which a scheduler then chooses a subset.

3) *Form remaining duties*

   The scheduler can also split up some straight runs generated in the first step and combine them with the remaining spells. The system provides little help to the scheduler beyond calculating the cost of the candidate duties.

The goal of this system is just to aid human schedulers to compile a schedule. It only builds a set of potential valid duties for schedulers to select from and further compile a schedule.

Later Elias attempted to develop an automatic driver scheduling system (Elias, 1966). He first developed a mathematical model, but found that the scheduling problem was too large to be solved by the model. He then turned to heuristic approaches instead.

The heuristic approach includes the following steps:

1) *Cut each block into a set of spells*

   The first pieces of work in each block are cut into a spell with a duration of approximate 3, 4 or 5 hours. Then the rest of the block is partitioned into spells of about 3 hours each or a maximum length if it is less than 3 hours.

2) *Form all possible 2-spell duties based on the set of spells*

3) *Construct many different schedules by selecting a different starting duty for each schedule*

   All the duties are ranked in order of cost, and at each stage the cheapest remaining duty that does not conflict with those already selected is added into the schedule until no more duties can be added. The spells of work not covered are compiled into single-spell duties.

4) *Output the cheapest schedule*.

The method is generate-and-select based, which involves generating a large set of potential duties from which a subset, constituting a solution, is selected. In Elias' method the duty generation process could only generate duties with up to two-spells. The duty selection process employed a greedy heuristic, which could obtain a solution quickly but the quality of the solution would usually be unsatisfactory. Although this system was only used on a test basis, the idea underlying this generate-and-select method may have influenced the development of other driver scheduling systems, in which the selection process employs more powerful approaches, e.g. Integer Linear Programming and meta-heuristics.

2.2.2 Weaver and Wren's work

Experience of using heuristics for driver scheduling at the University of Leeds started in 1967 (Wren, 1968; Weaver, 1968; Weaver and Wren, 1970 and 1972; Weaver, 1972; Manington and Wren, 1975). At the early days, it was hoped that heuristics could be developed which could improve any feasible schedule to a good final schedule. However, it became evident that the refining techniques were not sufficiently powerful to convert a poor initial schedule into a good final schedule, and they were frequently trapped in local optima. Therefore, the major effort was turned to finding a good initial schedule while the optimisation programs were still applied. This work led to the TRACS system (see Section 2.2.3) in the 1970's.

The first attempt of Weaver (1968) was to develop a driver scheduling program using the labour agreement in Leeds City Transport. The program only used a typical route of 33 buses from the Leeds schedule, and the route had only one relief point. Weaver claimed that the program would easily be extended to a route with more than one relief point. This program first estimated the least number of driver duties that would be needed to cover all the work in the

schedule, and split all the work between the duties in some arbitrary way. This initial schedule usually was infeasible. Then, the program was divided into two parts, the first of which aimed to produce a feasible schedule, and the second of which aimed to reduce the total cost. A similar algorithm was used in the two parts of the program.

The pieces of work were considered in pairs, and they attempted to change between the two duties all the work which came after the two pieces under consideration. That was, each of the two duties was first divided into two at the end of the pieces of work being considered, and then, the first part of work in each duty would be linked by the second part of work in the other duty. This operation was done only when the total infeasibility or the total cost was reduced. If the infeasibility could not be removed after no improvement could be made by the first part of the program, another duty would be introduced and the process was repeated.

Weaver (1968) claimed that this algorithm had two main disadvantages. One of these was that this algorithm was time-consuming, and the other was that the algorithm had the tendency to split up a continuous stretch of work into several short pieces of work.

Later, Weaver and Wren (1970 and 1972) modified the approach to build a good feasible initial schedule first and then to refine the schedule. Only limited tests were carried out on the approach, and no better schedules than the manual schedules were reported. Weaver (1972) summarised that the program was likely to produce good results if there was flexibility in both data (vehicle work) and labour agreements, and less good results otherwise. This work was superseded by TRACS.

2.2.3 The TRACS system

The TRACS (Technique for Running Automatic Crew Scheduling) system (Parker and Smith, 1981) was developed in the University of Leeds in the 1970's based on the work by Weaver and Wren (1970, 1972). It consisted of an initial program and a series of optimisation programs. The initial program constructed a good initial schedule, while the optimisation programs refined it. TRACS aimed to form a good initial schedule with the assumption that the refining process was unlikely to transform a poor initial schedule into a good one. An initial schedule is constructed as follows:

1) Early duties are constructed, which satisfy all the constraints.

   First mark the latest relief time, by which the driver who takes the bus out of the depot must be relieved for a meal break. Then the bus with the earliest remaining marked time is considered to form a duty by taking over another bus at or before its marked time. The marked time that must be used is said to be fixed when it is at the end of the bus work or it has already been selected to start the second part of a duty for another driver. If a marked time is not fixed, an additional driver will have to take over the bus at that time. It is better to have the additional drive starting as early as possible so that the marked time may be fixed earlier.

   Throughout the process of forming early duties, sufficient first spells of work are left unallocated to provide the desired number of split duties.

2) Late and middle duties are constructed by a similar process, but working backwards from the end of the day into the afternoon. Evening peak parts of the bus work that are suitable for the second part of split duties are not allocated.

3) Split duties are constructed by coupling the first and second halves of split duties.

4) Any remaining morning peak work is assigned to an extra early duty while the other unallocated work is attached to existing duties if possible.

The refinement process contains two sets of routines. One set attempts to reduce the number of duties or unallocated spells of work. Each duty is considered in turn to determine whether the work in it can be redistributed among the other duties. The duties with longer spreadovers are extended to be allocated more work, while the shorter duties become shorter and are hopefully removed altogether. Unassigned spells of work are reduced in length if possible. Another set of routines attempts to reduce the cost of the schedule, including any notional cost of undesirability. There are several routines to achieve this. One is to split up the duties at the meal break and to re-combine them as an assignment problem. The others are to exchange the links between spells if the cost is reduced.

The system was successfully used to compile schedules for several different bus companies. Since the heuristics used for constructing an initial schedule were sophisticated and customised for a specific problem, it could fully meet specific requirements but it usually took several months to adapt the heuristics to a new problem.

The TRACS approach assumed that duties with long spreadovers were ‘good’ duties. Hence, it tried to form long spreadover duties as far as possible. After compiling such ‘good’ duties, there

would be short spells of work remaining and the schedule might be inefficient as a whole. While many early heuristics could only compile duties with up to two-spells, TRACS could generate the duties containing two or three spells. However, the generation of multi-spell duties was dependent of the refinement process, which did not seem to have a means of escaping from local optima.

2.2.4 RUCUS and RUCUS II

RUCUS (RUn CUtting and Scheduling) (Wilhelm, 1975) is a bus and crew scheduling system developed by the MITRE Corporation under the sponsorship of the U.S. Urban Mass Transportation Administration in the early 1970's. This system was installed in a number of bus companies in the USA and Canada from 1975.

RUCUS had many similarities with TRACS, which constructed an initial schedule and then heuristically refined it. In the process of the construction of an initial schedule, RUCUS first formed one-spell straight duties without a mealbreak, and then formed two-spell early and late straight duties with an upper limit on the length of the intervening mealbreak, and two-spell early and late duties with mealbreaks of any length. Finally any unallocated work was added to the schedule as 'trippers', i.e. short one-spell duties.

Besides labour agreement rules, RUCUS required some soft rules to help cut the blocks. Different soft rules affected significantly the quality of the initial schedule. Therefore sometimes the system was run several times with different soft rules to get an acceptable initial schedule.

Having formed an initial schedule, several improvement algorithms were applied to eliminate the trippers by incorporating them into two-spell duties, and to reduce the cost of the schedule by shifting the spells between pairs of duties and by adjusting duties to use the preceding or succeeding relief opportunities. These algorithms were called *shifts* and *switches* respectively.

RUCUS was field-tested in 1972 and 1973, and released to the industry in 1974 (Schmidt and Knight, 1981). The system was found hard to control and considerable experimentation with the parameters was required to get a good schedule. Since its launch, there were many modifications made by consulting firms and users at various transit authorities, which led to several versions of RUCUS. SAGE's interactive package was one derivation (Hildyard and Wallis, 1981). SAGE did not modify the RUCUS algorithms. It just added natural language interfaces at different stages of system, which allowed schedulers to set parameter values between stages.

Later, RUCUS was substantially revised (Luedtke, 1985). The new generation of RUCUS, called RUCUS II, was released in 1982. The main change was similar to that made in SAGE, which made it easier to specify and modify parameters during the process of forming the schedule. The scheduling process was interactive instead of operating in batch mode.

Ball et al (1985) enhanced RUCUS II by replacing the shifts and switches algorithms with a graph-matching algorithm. The enhanced RUCUS II was applied to two test problems and better results were produced.

The RUCUS II system only automatically formed duties with up to two-spells. It was mainly designed for North American operators and might not be good for European operators. This system has now been abandoned.

2.2.5 The COMPACS system

COMPACS (COMPuter Assisted Crew Scheduling) is an interactive system developed in the early 1980's by a commercial company (Wren et al., 1985). It originated in an interactive crew scheduling system called TRICS (Techniques for Running Interactive Crew Scheduling), which was developed at the University of Leeds in the late 1970s. COMPACS also incorporated some heuristics used in TRACS for generating an initial schedule, but not the refinement programs. In the earlier 1980's it was incorporated into the BUSMAN sche duling package (Chamberlain and Wren, 1992).

COMPACS, as an interactive system, could help schedulers in a range of ways. It allowed the schedulers to build up the schedule one or more duties at a time; at any stage, the duties already formed could be unassigned or modified. COMPACS could also form a complete schedule automatically, but the quality of schedules was generally poor because it did not have any refining functions.

2.2.6 Overview of the early heuristic approaches

The heuristic systems were successful on some applications. The reason is that they were customised for individual companies; hence, they could be fully tailored to meet the company's requirement. However, they were heavily reliant on domain knowledge and specific company needs and therefore they were not easily portable to other companies and had to be considerably modified in order to fit new conditions.

By the 1980's it was generally recognised that heuristics alone were not suitable for general use (Wren and Rousseau, 1995). The purely heuristic approach was therefore abandoned in favour of mathematical programming approaches aided by heuristics.

2.3 Mathematical programming approaches

In the late 1970s, research in driver scheduling methods has stepped into the second period, in which mathematical programming approaches were popular. Most currently successful driver scheduling systems are based on mathematical programming approaches, in which the driver scheduling problem is commonly modelled as either a set covering or a set partitioning problem. In this section, three systems IMPACS, TRACS II and HASTUS are reviewed. TRACS II and HASTUS are probably the most successful driver scheduling systems in the world. IMPACS is the precursor of TRACS II. The TRACS II system will be used for benchmarking results in this research.

2.3.1 Mathematical model of set covering and set partitioning

The following description of the *set covering problem* is taken from Reeves (1996).

> *A family of m subsets collectively contains n items such that subset* S_i *contains* n_i ($\leq n$) *items. Select k* ($\leq m$) *subsets* $\{S_{i_1}, S_{i_2} \ldots, S_{i_k}\}$ *such that* $| \bigcup_{j=1}^{k} S_{i_j} | = n$ *so as to minimise*
>
> $$\sum_{j=1}^{k} c_{i_j}$$
>
> *where* c_i *is the cost of selecting subset* S_i.
>
> *Here the solution is represented by the family of subsets* $\{S_{i_1}, S_{i_2} \ldots, S_{i_k}\}$.

If, in addition, the subsets $\{S_{i_1}, S_{i_2} \ldots, S_{i_k}\}$ are mutually exclusive, the above problem becomes a set partitioning problem.

The driver scheduling problem with n pieces of work and m previously generated duties can be modelled as a set covering problem and expressed as the Integer Linear Programming (ILP) problem:

$$\text{Minimise} \quad \sum_{j=1}^{m} c_j x_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{j=1}^{m} a_{ij} x_j \geq 1 \quad \text{for } i = 1,2,\ldots,n \tag{2.2}$$

where,

$$x_j = \begin{cases} 1 & \text{if duty } j \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{ij} = \begin{cases} 1 & \text{if duty } j \text{ covers piece of work } i \\ 0 & \text{otherwise} \end{cases}$$

c_j is the cost associated with the j^{th} duty.

By changing the constraint (2.2) into the following form, it becomes a *set partitioning* problem:

$$\text{subject to} \quad \sum_{j=1}^{m} a_{ij} x_j = 1 \quad \text{for i} = 1,2,\ldots,n \tag{2.3}$$

In the set covering model, each piece of work must be covered by at least one duty; in a set partitioning model, each piece of work must be covered by exactly one duty. In theory, the solution to the ILP is the schedule with minimum cost. However, in practice, the total number of all possible duties is usually very large, and the resulting ILP cannot be practically solved. The very large potential duty set therefore has to be reduced to a manageable size by heuristics

or some other means, or else the problem has to be decomposed into several sub-problems and solved separately.

2.3.2 The IMPACS system

IMPACS (Integer Mathematical Programming for Automatic Crew Scheduling) is a driver scheduling system developed in University of Leeds in the late 1970's. It was originally developed for bus operation. Parker and Smith (1981) presented the prototype version and the full description of the system was described by Smith (1986 and 1988), and Wren and Smith (1988).

IMPACS is based on a set covering model. As mentioned above, the number of variables and constraints can be enormous; reduction heuristics are therefore employed by IMPACS. The system solves the driver scheduling problem by the following steps:

1) *Add some soft rules*

   The number of variables, i.e. the number of candidate duties considered, depends on the rules governing the legality of a duty. The more tightly set rules are applied, the less possible legal duties are formed. IMPACS adds some extra rules, known as *soft rules*, such as the minimum length of spells, and the minimum work content, etc. The soft rules, whose parameters are usually specified by the scheduler, prevent generating inefficient duties, and duties that are unlikely to contribute to forming a good schedule.

2) *Reduce the number of pieces of bus work using heuristics*

   There are two optional modules in this step. EST identifies a list of relief opportunities that may be critical to forming a good schedule. SELECT analyses the remaining relief

opportunities and de-selects those not likely to be used, thereby reducing the number of pieces of work.

3) *Generate a large set of legal duties according to the labour agreement rules and the soft rules using heuristics*

4) *Reduce the number of duties using heuristics*

   Some duties, which are considered less crucial (e.g. every piece of work in the duties is covered by a specified number of other duties) than the others for forming a good schedule, are eliminated from the large duty set.

5) *Select a subset of duties using ILP*

   The driver scheduling problem is now expressed as a set covering problem. IMPACS solves this problem by first obtaining a relaxed LP solution and then looking for an integer solution using a Branch-and-Bound (B&B) algorithm. The B&B algorithm searches for integer solutions through a tree structure. The search terminates when a sufficiently good solution is found or a certain pre-defined search depth has been reached.

IMPACS also provides a decomposition module for solving large problems. After decomposition and the first sub-problem has been solved, the work of inefficient duties in the schedule are carried forward to the next sub-problem. Finally, the results of all sub-problems are combined into a complete schedule.

In 1990, IMPACS was adapted to estimate the effects of changing train driver scheduling requirements (Parker et al., 1995). Since train driver scheduling problems are usually larger and more complicated than bus driver scheduling problems, it is difficult to apply the IMPACS

system, which was designed for bus driver scheduling problems. IMPACS is now superseded by TRACS II.

2.3.3 The TRACS II system

TRACS II (Techniques for Running Automatic Crew Schedules, Mark II) has been developed since 1994 specifically to satisfy the needs of rail operations, while still being capable of producing bus driver schedules. TRACS II follows almost the same approach as IMPACS, but the components have largely been redesigned to cope with the complexity of rail operations and to incorporate new algorithmic advances.

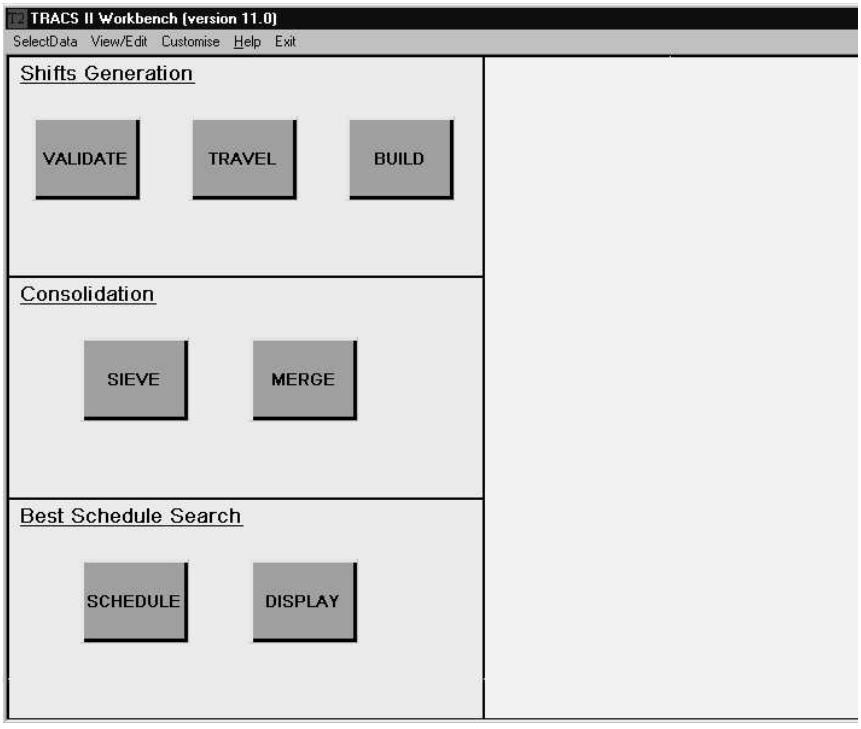


Figure 2.1 Workbench main window of TRACS II

From Figure 2.1, it can be seen that TRACS II consists of seven modules in three stages. The first stage is called *shift generation*, which constructs many potential duties. VALIDATE

checks first the validity of data sets. Then TRAVEL calculates all possible opportunities for drivers to travel as passengers on scheduled services or taxis between relief opportunities. BUILD finally generates a large set of possible valid duties (a set of filtering parameters is used to limit the generation of too many duties). The heuristics in BUILD are different from IMPACS. The second stage contains a module called SIEVE, which ranks the duties generated by BUILD and eliminates those deemed to be inferior. The set of duties generated by BUILD is trimmed down to a desired size before it is passed to the mathematical solver called SCHEDULE. Recently, SIEVE is often only used to remove duplicated duties because the integration of a column generation method into SCHEDULE now allows it to accept larger duty sets. MERGE is only used when BUILD is run more than once. It merges sets of duties generated according to different parameters or different decomposed sub-problems. The mathematical process called SCHEDULE is the main module in the third stage, which is originated from the ILP process used in IMPACS. IMPACS could handle up to 30,000 input duties, but there is virtually no limit for TRACS II. SCHEDULE has two main choices of optimisation routines, which are that of a primal column generation approach and a dual steepest edge approach. Fores et al (2001) recommended that the dual steepest edge approach be executed on problems containing at most 30,000 duties, where all duties can be explicitly considered within the limits imposed for storage and efficiency. However, users can select the column generation approach to run any size of problem. SCHEDULE consists of four processes. The first process builds an initial integer solution using heuristics if column generation is selected as the solution algorithm. This process also selects an initial duty set of up to 30,000 duties from the large set of potential duties generated by BUILD. The second process uses a set covering model and obtains a LP relaxation. The second process will be repeated if it is necessary to add new columns of duties. The third process called REDUCE is introduced by Smith and Wren (1988) and enhanced by Willers (1995). It selects the duties that start or end at an RO that is used in the LP solution. The other duties are removed and pieces of

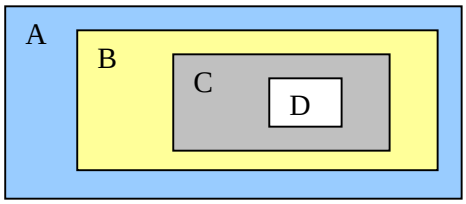
work not using a selected RO are combined. This is an optional process, but reduction in problem size decreases solution time and does not seem to inhibit the quality of the schedule. The fourth process searches for an integer solution using a Branch and Bound method. If a solution is found, DISPLAY will print the schedule in a condensed format.

TRACS II has been successfully installed in several transport companies. Over the period 1995-1996 TRACS II satisfactorily produced schedules for about twelve of the privatised British train companies (Kwan et al., 1999). It is now being installed in the 26 companies of First Group, the largest bus company in the UK.

TRACS II is also used for benchmarking by a number of researches in Leeds University. Parts of the system are utilised in some other driver scheduling approaches, e.g. Forsyth's ant system, Kwan's and Li's genetic algorithms, Li's simulated evolution algorithm, and La yfield's and Curtis's constraint programming approaches (see Sections 2.4 and 2.5).

Since TRACS II uses a set covering model, the schedule generated may (and usually) contains overlapping driving work (over-cover), which creates unproductive time for drivers. Over cover is eliminated manually before a schedule is operated.

Also, because of the use of ILP, a series of techniques to reduce the number of variables (potential duties) and constraints (ROs) have to be operated according to the capability of the ILP solver. Figures 2.2 and 2.3 illustrate the processes of reduction, where A, B, C, D denote the sets of relief opportunities at different levels, while E, F, G, H, I denote the sets of duties at different levels. The reduction at each level decreases the possibility of producing an optimal solution.



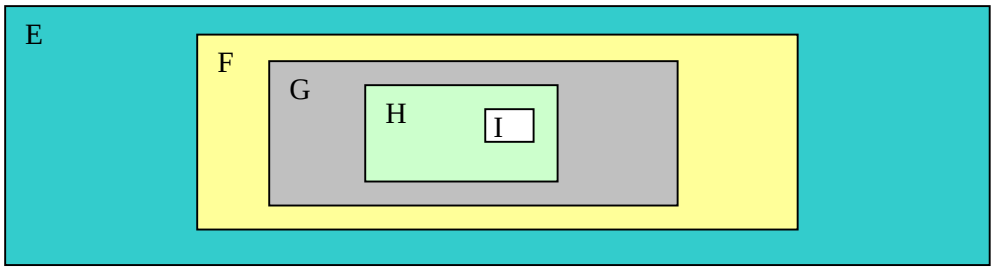
A: Full set of windows of relief opportunities.

B: Reduced set of relief opportunities after shrinking a WRO into one or two selected ROs.

C: Reduced set of relief opportunities after removing the ROs not used in LP relaxation. (*This is optional so might not be reduced*).

D: Set of active relief opportunities.

Figure 2.2 Different levels of RO reduction



E: Set of all possible valid duties according to labour agreement rules

F: Set of all possible valid duties based on tightened governing rules and the reduced set of ROs in level B, i.e. not considering windows of relief opportunities.

G: Reduced set of potential duties selected by the column generation method.

H: Reduced set of potential duties selected based on the set of ROs in level C.

I: Set of duties in a schedule, which may contain overlapping duties.

Figure 2.3 Different levels of potential duty reduction

2.3.4 The Crew-opt method of the HASTUS system

HASTUS is a commercial package that deals with all the scheduling issues: bus vehicle scheduling, driver scheduling, and rostering. It was originally developed by the University of

Montreal in the 1970's and later jointly with Giro Inc., Canada. The work on the HASTUS system has been reported in a number of papers (e.g. Lessard et al., 1981; Carraresi and Rousseau, 1982; Rousseau et al., 1985; Rousseau and Blais, 1985; Blais and Rousseau, 1988). The latest version of HASTUS was exhibited in the international conference on Computer-Aided Transit Scheduling held in June 2000, Berlin. A concise description is provided in Section 2.6.

A crew scheduling method called Crew-opt is part of the HASTUS system (see Desrochers and Soumis, 1988, 1989; Desrochers et al., 1992; Rousseau and Desrochers, 1995), which has been used in a wide range of problems including a train company (East Japan Railway). The Crew-opt approach solves the driver scheduling problem using a set covering (or set partitioning) method incorporating a column generation technique.

Crew-opt generates a large set of legal potential duties based on a set of 'useful' ROs from which a set of 'good' duties is heuristically selected. It then solves the driver scheduling problem modelled as a set covering/partitioning problem with LP. The LP process will be repeated if new columns of interest are added. After this has been done, a Branch and Bound method is used to find an integer solution.

Similar to TRACS II, HASTUS is an ILP-based approach, which selects a subset from the set of duties generated. The difference is that HASTUS does not generate all possible legal duties beforehand but a 'good' set of potential duties instead. After each LP solution, the dual variables are used to generate new duties with negative reduced costs susceptible to improving the current LP solution. The column generation method in HASTUS is used to generate new columns of duties; however, in TRACS II it is used to select new columns of duties from the previously generated set.

Despite the use of column generation, filtering labour rule parameters are still employed to eliminate 'useless' relief opportunities and to limit the set of feasible duties generated. These parameters might guide the duty generation throughout the scheduling process. The column generation process might not be able to generate the duties with newly introduced constraints because it is based on the LP relaxation. This approach solves a number of LP problems until it becomes impossible to find a feasible duty satisfying its criteria (i.e. with a negative reduced cost). The whole process is likely to be time-consuming.

2.3.5 Overview of the mathematical programming approaches

The mathematical approaches based on a set covering or a set partitioning method have been successfully used in many practical systems with the aid of heuristics. However, optimal solutions still cannot be ensured and the systems cannot be treated as black boxes that produce working schedules. The limitations might be summarised as follows:

- More easily adapted to different problems than early heuristics, but some adaptation is still needed sometimes, and this adaptation may be considerable;
- Filtering rules are often applied so that the set of potential duties generated would not be prohibitively large, so the optimality of solutions would be compromised. The manipulation of the parameterised filtering rules may appear obscure to schedulers not versed in scientific disciplines;
- Generation of all possible legal duties may be very time consuming if slack scheduling rule parameters are used;
- Some reduction on problem size or in the scheduling process is inevitable;

- It is likely that there will always be problems for which decomposition is necessary on size grounds;
- It is possible that for some problem instances no solution can be found within a reasonable time;
- Windows of relief opportunities cannot be handled practically.

2.4 Meta-heuristic approaches

While the mathematical methods continue to be improved, several strands of research into meta-heuristics for driver scheduling have begun in the 1990's.

2.4.1 A Tabu Search approach - Cavique et al

Cavique et al (1999) have presented a Tabu Search approach for crew scheduling. The approach constructs an initial schedule using a method similar to that used by TRACS, which is called a *run-cutting algorithm* by Cavique et al (1999). The initial schedule is feasible according to a set of rules, and covers a pre-defined timetable. *Tabu-crew*, a refining algorithm embedded in a Tabu Search framework, is applied to reduce the number of duties. *Tabu-crew* iteratively removes some 'inefficient duties' (with only a single spell of work) as well as their 'adjacent' duties from the current solution, and then applies the run-cutting algorithm to construct duties to cover the broken schedule. The 'adjacent' duty of an inefficient duty was not defined in (Cavique et al., 1999). A frequency-based memory was used. The frequency values indicated the number of times that each trip was assigned to a single-spell duty. Two-spell duties containing trips with high frequencies, were evaluated favourably. This strategy provided 'difficult trips' more chances to be combined with others at an early stage of the algorithm. Cavique et al (1999) claimed that this algorithm was very efficient at improving the initial

solution after a few iterations, but thereafter it took a significant amount of time to find a better solution.

Cavique et al (1999) also presents another Tabu Search algorithm called *Run-ejection*, which considers compound moves based on a *subgraph ejection chain method*. Ejection chain (Glover, 1996) is a type of approach in Tabu Search for creating compound moves. It results from a succession of steps in which an element is assigned to a new state, with the outcome of ejecting some other element from its current state. The ejected element is then assigned to a new state, in turn ejecting another element, and so forth, creating a chain of such operations. The *Run-ejection* algorithm first divides each block of vehicle work into feasible spells of work and creates the node set P containing all spells. It then links pairs of spells building all possible duties and forms the edge set $D = \{(p_i, p_j) | p_i, p_j \in P\}$. The matching graph $G = (P, D)$ is therefore built. The left spells form single-spell duties, which are unmatched nodes. Since there is a large number of possible graphs G, a tabu search framework with compound moves is used to explore this enormous number of alternatives. The compound moves contain three elementary operations: 'shift operation' which shifts the endings of a spell to the right or to the left; 'cut operation' which cuts one spell into two spells; and 'merge operation' which combines two spells into a single spell. For each block divided as above, these three operations are considered in sequence and constitute an *ejection chain*. The matching graph is revised after each operation, during which some new feasible edges are built while the infeasible edges caused by the operation are removed. A feasible solution can be obtained at each step of the ejection chain process.

The tabu search algorithms were designed as a part of a decision support system for the crew scheduling management of the Lisbon Underground. Cavique et al (1999) only evaluated the algorithms using data provided by Lisbon Underground. The computational results represent a

reduction of up to 3.6% (in terms of number of duties) on the manual solutions produced by the Lisbon Underground planners, and the Run-ejection algorithm is superior to the Tabu-crew algorithm.

These algorithms only construct one- or two-spell duties. The complexity will increase dramatically when duties with three or more spells are allowed. Duty costs are not considered, the objective is only to minimise the number of duties. Whereas these restrictions may be fine for Lisbon Underground, they may not be satisfactory for other bus or rail operations.

2.4.2 Genetic Algorithms

Genetic Algorithm (GA) is originated from Holland (1975) for solving hard combinatorial optimisation problems. It simulates the genetic evolution process to produce a 'fit' (high quality) solution, in which the fitter individuals have a better chance of survival while the weaker individuals are replaced by new generations. A chromosome is presented as a string of genes defining the characteristics of a solution to a problem. GA improves the current population by replacing weaker parents with fitter offsprings by the operation of selection, crossover and mutation.

2.4.2.1 Kwan et al

The approach presented in Kwan et al (1999a) and Kwan (1999) uses a Genetic Algorithm approach to schedule bus and train drivers. This work built on experience of the earlier attempt by Wren and Wren (1995). The main purpose of this approach is to replace the Branch and Bound method in the ILP process in TRACS II. The potential duties as well as the relaxed LP solution produced by TRACS II are used. The GA process only represents in the chromosomes

preferred duties, whose fractional values in the relaxed LP solution generated by TRACS II are higher than a pre-defined constant, e.g. 0.2. The reason why only these duties are chosen as genes is that Kwan (1999) finds by experiments that an average of 74% of the duties in the best integer solutions obtained by TRACS II have fractional values higher than 0.2 in the corresponding relaxed solutions. These preferred duties form the backbone of a schedule. A solution set is completed by using a greedy repair technique.

For some problems the approach has produced schedules as good as TRACS II solutions in terms of number of duties. In some other cases, one or two more duties are needed. For one large problem instance, which was decomposed by TRACS II at the time of the experiment, the number of duties is reduced, but the schedule is more expensive in terms of total cost.

2.4.2.2 Li and Kwan

Li and Kwan (2000) have presented a genetic algorithm with fuzzy comprehensive evaluation for driver scheduling. This research has the same objective as that of Kwan et al presented in the previous section. Both are intended to replace the Branch and Bound process in TRACS II. Similarly, Li and Kwan also make use of the potential duties and the LP relaxation solution produced by TRACS II. The difference is that a genetic algorithm is employed in this work to produce a near-optimal weight distribution amongst fuzzified criteria of evaluating individual potential duties instead of finding a schedule directly. The fuzzified criteria are expressed as five fuzzy membership functions, the weights of which are encoded as a chromosome. Each chromosome is a 30-bit string divided into six bits per weight. A greedy heuristic makes use of the fuzzy comprehensive evaluation to select duties to construct a schedule, the total cost of which is used to evaluate the fitness of the chromosome in the evolutionary process. The best schedule results as a by-product.

This approach has been tested on eleven real-world problems and the results are comparable to the best known solutions.

2.4.3 A Simulated Evolution algorithm – Li and Kwan

The latest research of Li and Kwan (2001) is on applying a Simulated Evolution (SE) algorithm for the driver scheduling problem modelled as a set covering problem. A SE algorithm is a general optimisation technique originally proposed by Kling and Banerjee (1987) based on analogy with the natural selection process in biological environments. The biological solution to the adaptation process is the evolution from one generation to the next one by eliminating inferior components and keeping superior ones for subsequent states. SE contains four main steps: *initialisation*, *evaluation*, *selection* and *reconstruction*. The first step initialises some parameters, e.g. stopping condition and a valid starting solution. The following three steps are applied iteratively. *Evaluation* calculates the goodness for each component of the current solution. *Selection* probabilistically selects components. Components with low goodness have a higher probability of being discarded. *Reconstruction* applies special heuristics to derive a new and complete solution from the former partial solution. Then the three steps are repeated until the stopping condition is reached. Throughout these iterations, the best solution is retained and finally returned as the final solution.

In the SE algorithm for driver scheduling, the steps of Evaluation and Reconstruction have been fuzzified. Having generated an initial solution using a greedy method, the Evaluation step evaluates each duty in the solution based on its coverage status and five fuzzified criteria similar to those in their previous work (see Section 2.4.2.2). These criteria are represented by fuzzy membership functions, which would lead to a quantitative formulation of the structural

goodness of this duty by the method of fuzzy comprehensive evaluation. The Reconstruction step applies a greed-based heuristic with the above derived fuzzy evaluation function to form a complete solution from a partial solution.

This is the first time that the SE algorithm has been applied to the driver scheduling problem. The experimental results are better than the previous solutions generated by the Genetic Algorithm but generally not as good as the TRACS II solutions.

2.4.4 An Ant System - Forsyth and Wren

Forsyth and Wren (1997) have applied a combinatorial optimisation method called *Ant System* for driver scheduling. Ant system was developed by Dorigo et al (1996) from the study of the behaviour of ants searching for food. The basic idea is that ants leave pheromone trails that can be detected by other ants. When ants set off to look for food, they move at random. Once food is found, the ant returns to the nest with highest probability along its own pheromone trail. The ants which find the shortest path are likely to return back soonest and their pheromone trails are therefore strongest. Subsequent ants are most likely to follow the shortest path because the ants have an in-built bias towards following strong pheromone trails. Therefore the shortest path will be strengthened more and more and it becomes the dominant trail.

The ant system for driver scheduling defines a network of nodes based on the potential duties generated by TRACS II. Each ant creates a solution by traversing the network from a selected relief opportunity. Relief opportunities are selected by a probabilistic heuristic and then the ant chooses a duty (node) from the set that starts at the relief opportunity. This is repeated until all the work is covered. The quality of solution corresponds to the pheromone level. As the process progresses, the old trails slowly evaporate over time unless the trail is visited by more ants. The

strongest pheromone trail represents the best solution. Unfortunately, this system does not generate schedules comparable to the TRACS II solutions.

2.4.5 Overview of the meta-heuristic approaches

The tabu search approach by Cavique et al was developed for a particular problem and was not able to compile duties with three or more spells. It might not be easily adaptable to different bus or rail operations. Amongst the other approaches reviewed, the evolutionary algorithms by Kwan et al and by Li and Kwan are promising; however they take advantage of TRACS II to obtain a large set of potential duties and the relaxed LP solution. They at best can only replace the final phase of TRACS II. Therefore they inherit parts of the drawback of the mathematical approach. As reported by Kwan et al (2000) and Kwan (1999), a pure genetic algorithm without using the relaxed LP solution did not produce acceptable solutions except for very small problems.

2.5 Constraint programming approaches

Constraint programming approaches for solving combinatorial optimisation problems are becoming widely used. They provide a powerful and easy system for modelling restrictions and use these restrictions to search for a solution. In recent years, attempts have been made to use them for solving driver scheduling problems.

2.5.1 Layfield et al

Layfield et al (1999) used constraint programming to produce a component that could be slot into the TRACS II system before building all potential duties. The aim of the work is to reduce

the number of relief opportunities such that the problem size is cut down. This has been done by removing the relief opportunities that are unlikely to be used in good schedules. This program initially works on forming the morning part of the schedule imitating the manual scheduling process. The formed part of the schedule contains efficient handing over between drivers (good 'mealbreak chains'), in which Layfield et al suggested that potentially useful relief opportunities existed. Several morning schedules are constructed, then the relief opportunities that are not used in these schedules are removed. This process can also be applied to the evening part of the schedule. A constraint programming tool ILOG Solver is used to solve the problem formulated as a constraint satisfaction problem.

2.5.2 Curtis et al

Curtis et al (1999) presented a constraint programming approach for forming bus driver schedules. They modelled the bus driver scheduling problem as a constraint satisfaction problem, where variables were defined as pieces of work and the domain of each variable was the set of indices of the duties that covered the piece of work. They then solved this constraint satisfaction problem as a set partitioning problem. This work also benefited from the TRACS II duty generation phase to obtain a large set of potential duties. The relaxed LP solution generated by TRACS II was used as a 'variable ordering' guide. The approach was tested on relatively small problems and produced schedules with the same number of duties as those in TRACS II solutions.

Later, Curtis et al (2000) provided an iterative repair process for constructing driver schedules. A local search method was used to solve the problem modelled as a constraint satisfaction problem. A comprehensive description is presented in Curtis (2000).

2.5.3 Overview of the constraint programming approaches

Two constraint programming approaches for bus driver scheduling have been reviewed in this section. Only the Curtis et al's approach can produce full bus driver schedules. Their approach is heavily dependent of the use of the relaxed LP solutions to guide variable and value ordering. Therefore, it inherits the limitation of mathematical programming approaches on solving large problem instances.

2.6 Recent transport scheduling packages

The latest international conference on computer-aid scheduling of public transport was held in June 2000 in Berlin, Germany. During the conference, several transport scheduling packages were exhibited. The following are some examples:

- GIST

GIST (Dias et al., 2000; Lourenço et al., 2000) is a decision support system to assist the planning department of transportation companies or transit authorities in operations management. GIST is designed as a modular system, including the following processes: network information management, route information management, timetabling management, vehicle scheduling, crew scheduling, and crew rostering. GIST gives special attention to interactive facilities and friendly interfaces, combined with optimisation tools.

GIST models the crew scheduling problem as a single objective linear programming problem to minimise the cost. Lourenço et al (2000) indicate that the bus companies involved in the GIST project have not been satisfied with the bus-driver module based on the single-objective LP,

even though the optimal solution could be obtained. Therefore, Lourenço et al (2000) present a multi-objective meta-heuristic approach for the bus-driver scheduling problem. Lourenço's approach aims to solve the driver scheduling problem in a user-friendly environment. The approach has been incorporated in GIST and can obtain many alternative solutions in a short time, but the detail has not been published yet.

- Trapeze

Trapeze (see http://www.trapezesoftware.com/index.html) is a commercial package of Trapeze Software Group, Canada. It provides transport software for public transport, dial-a-ride, student transport, medical transport, and other community transport services. It includes a function for scheduling public transport drivers. In the exhibition, a demonstration of driver scheduling was given. Trapeze produced a solution very quickly but the quality of the solution was difficult to judge and no information on the technique used was made available.

- HASTUS

HASTUS (see http://www.giro.ca/) is the result of over 20 years of ongoing research and development at GIRO, in collaboration with the University of Montreal's Centre for Research on Transportation. The latest version of HASTUS, called *HASTUS 5*, offers an integrated transit database and a set of application modules, e.g., *HASTUS-Vehicle*, which generates vehicle timetables and vehicle schedules; *HASTUS-Crew*, which produces crew schedules, and *HASTUS-Roster*, which automatically generates multi-day operator assignment. The approach used in the crew scheduling component is described in (Rousseau and Desrochers, 1995) (see Section 2.3.4).

2.7 Conclusions - generate-and-select approach vs. constructive approach

This chapter has reviewed the driver scheduling approaches, which can be roughly divided into two groups: *generate-and-select approach* and *constructive approach*.

The generate-and-select approach is at present the most successful for bus and train driver scheduling. It involves generating a set of legal potential duties from which a minimal and most efficient subset is selected. The generation process is problem-oriented while the selection process can be algorithm-oriented. After generating a set of duties, a solution may be found by a variety of selection methods, such as mathematical programming, evolutionary algorithms, constraint programming, etc. This makes the approach adaptable to different situations although the generation algorithm is likely to be adjusted depending on the situations. Most approaches used in present commercial systems such as TRACS II and HASTUS can be regarded or implicitly regarded as generate-and-select approaches. The recent researches of Kwan et al, Li and Kwan, and Curtis et al are also based on the generate-and-select approach (see Sections 2.4 and 2.5). Unfortunately, the number of potential duties is usually enormous, which precludes the selection methods from finding an optimal solution in practical time. It is therefore inevitable that some restrictions have to be imposed to limit the set of potential duties to within an acceptable size or search depth, etc. Optimality is therefore compromised. Moreover, manipulation of the parameterised restrictions is usually hard for non-experienced users.

In contrast, a constructive approach, which constructs an initial schedule and then refines it iteratively, has no need of artificial rules or explicit reduction in problem size. Obviously for constructive approaches, the fewer rules that are imposed on the validity of a driver duty, the easier it is to construct a solution. Moreover, the more relief opportunities that are provided in

the blocks, the more chance it has to refine the schedule. Windows of relief opportunities provide ranges of opportunities for drivers to change over. The constructive approach could take benefit of windows of relief opportunities to refine schedules with more flexibility in recutting blocks. This opens up the opportunity of building a realistic model with a chance of obtaining 'good' solutions.

The early heuristic systems (see Section 2.2) are based on the constructive approach. Unfortunately, the refinement functions, where one was used, were very weak. They relied on good initial schedules constructed based on human schedulers' knowledge, such as in TRACS and RUCUS, and hoped to get a good final solution by simple refinement. This is the reason that the early heuristics tended to require much effort to be adapted to new situations. Also, extensive final manual adjustments to the schedule were needed. Possibly none of the early heuristics are still in use. Cavique et al's tabu search approach (see Section 2.4.1) is of a constructive type. The refining algorithms Cavique et al used are more powerful than those in the early heuristic systems because a tabu search technique is applied, which can help escape from local optima. However, Cavique et al's algorithms only construct duties with up to two spells, aim to minimise only the number of duties but not the cost, and are only evaluated for Lisbon Underground. The algorithms may not be suitable for most other practical situations.

The early constructive approaches have been abandoned. Instead, the generate-and-select approach is at present the most successful for bus and train driver scheduling. However, the advantages of constructive approaches can overcome some limitations inherent in generate-and-select approaches. Furthermore, the modern computing power and modern searching methods make good refining algorithms possible. Unfortunately, the solution quality of a constructive heuristic approach is difficult to guarantee.

The research presented in this thesis will investigate a constructive approach to tackle the bus and train driver scheduling problem with windows of relief opportunities, which the generate-and-select approach cannot handle. The initial research is on 2-opt heuristics presented in Chapter 3, based on which further research into a tabu search meta-heuristic approach is presented from Chapter 4 onwards.

Chapter Three

Neighbourhood Search Heuristics for Driver Scheduling

3.1 Introduction

Many hard combinatorial optimisation problems in areas such as resource allocation, packing and scheduling have traditionally been tackled using mathematical programming methods (Reeves, 1996). However, the mathematical framework restricts the form of the objective function and/or the constraints. The assumptions in such models are often only approximately true. Heuristic methods are usually more flexible and may have no need of such assumptions. Capturing the salient features of the heuristic methods Rayward-Smith et al (1996, p.5) gives a definition as follows:

> ''*A* heuristic technique (*or simply, a* heuristic) *is a method which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee optimality, and possibly not feasibility. Unfortunately, it may not even be possible to state how close to optimality a particular heuristic solution is.*''

From this definition it seems that there are severe problems inherent in the use of heuristics. However many heuristics, especially modern heuristics, do produce high-quality solutions in practice. They can obtain near optimal solutions for many difficult real-world problems with reasonable computational effort.

The driver scheduling problem is NP-hard (Wren, 1998). This research attempts to investigate two heuristic approaches for the driver scheduling problem. One is a *2-opt heuristic* (Lin, 1965), which is a simple Neighbourhood Search method. The other is a *Tabu Search* method (Glover and Laguna, 1997), which may be considered as an elaborate Neighbourhood Search method. A 2-opt heuristic approach is investigated in this chapter. A Tabu Search meta-heuristic approach is investigated in Chapters 4 to 6.

This chapter presents first the basic idea of the Neighbourhood Search methods, and then overviews some of the successful applications of 2-opt heuristics. Following this, a bus vehicle scheduling system using 2-opt heuristics is introduced while the complexity of the driver scheduling problem in applying the 2-opt heuristics is initially analysed. Some simplifications to the problem have been made and are presented in Section 3.4. The aims of the research are:

- to investigate the applicability of 2-opt heuristics for driver scheduling;
- to design the neighbourhoods, which are potentially suitable for the driver scheduling problem with WROs;

- to design the methods for escaping from local optima;
- to investigate the potential of the constructive heuristic approach.

The proposed 2-opt driver scheduling approach is outlined in Section 3.5 while its major components are depicted in Sections 3.6 to 3.9. The implementation and some computational results are presented in Section 3.10 before concluding remarks are given at the end of this chapter. Some of the work in this chapter has been presented by Shen and Kwan (2000).

3.2 Neighbourhood Search

A fundamental concept of the heuristic methodology is that of *neighbourhood search* (NS) (Aarts and Lenstra, 1997). Roughly speaking, a neighbourhood $N(x)$ of a solution x is a set of solutions that can be reached from x by an operation called a *move*. For example, in a sequencing problem, the interchange of two elements in a solution is a common move. If a solution x^* is better than any other solutions in its neighbourhood $N(x^*)$, then x^* is a local optimum with respect to this neighbourhood. Table 3.1 outlines a neighbourhood search method.

Table 3.1 Neighbourhood Search Method

Step 1	Initialisation (get an initial solution x^{now}, and record the current best solution by setting $x^{best} = x^{now}$).
Step 2	Choose a solution $x^{next} \in N(x^{now})$. If no solution qualifies to be x^{next} or other termination criteria apply, then the method stops.
Step 3	Set $x^{now} = x^{next}$, and if x^{now} is better than the current best solution, set $x^{best} = x^{now}$. Go to Step 2.

The definition of the best solution is normally dependent on the objective function (e.g. the cost of solution x), which is denoted by $c(x)$. The choice criteria for selecting moves and the termination criteria for ending the search are given by some external prescriptions. By specifying these prescriptions in different ways, a variety of procedures can be generated.

One of the most popular neighbourhood search methods for finding an approximation to the minimum cost is perhaps the *descent method* (Rayward-Smith et al., 1996). Descent Method only allows moves that improve the current solution. If no improving solutions can be found, the method terminates. Table 3.2 expresses the descent method by only showing Step 2 since Step 1 and 3 are the same as those in Table 3.1.

Table 3.2 Descent Method

Step 2	Choose a solution $x^{next} \in N(x^{now})$ such that $c(x^{next}) < c(x^{now})$ and terminate if no improving solutions can be found.

There are two popular ways to choose the improving solution. One selects the first encountered improving solution, while the other selects the best improving solution among the neighbourhood of the current solution. The former is called *Mild Descent Method,* while the latter is a greedy method called *Steepest Descent Method* and its Step 2 is expressed in Table 3.3 (Step 1 and Step 3 are the same as those in Table 3.1).

Table 3.3 Steepest Descent Method

Step 2	Choose a best solution $x^{next} \in N(x^{now})$ such that $c(x^{next}) \leq c(x)$ for any $x \in N(x^{now})$ and terminate if no such solution x^{next} exists.

Obviously the best solution generated by the descent method is only guaranteed to be a local optimum. Therefore this is also often referred to as *Local Search*. In many combinatorial problems there are usually many local optima while there are only one or few global optima. The chance of a local optimum being a global optimum is usually very low. Various methods or strategies are designed in applications to continue the search before or after being trapped at local optima to find an improved solution, for example, the *variable-depth search algorithms* introduced by Kernighan and Lin (1970) for the graph partitioning problem and for the travelling salesman problem (Lin and Kernighan, 1973). Other recent methods, often labelled as *meta-heuristics* (Rayward-Smith et al, 1996; Aarts and Lenstra, 1997), are designed to escape from the local optima traps. Generally, they still do not guarantee an optimal solution nor performance bounds. However, they are widely used in solving combinatorial optimisation problems and give good quality results for a wide variety of problems in practice (see Glover et al, 1993; Aarts and Lenstra, 1997).

3.3 2-opt heuristics

A 2-opt heuristic (Lin, 1965) is a kind of *Mild Descent* neighbourhood search method, in which an improved solution is obtained by exchanging two distinct components in the current solution. 2-opt heuristic is very simple but effective. This is why 2-opt is one of the most popular heuristic methods for symmetric travelling salesman problems (TSP) (see Lin, 1965; Lin and Kernighan, 1973; Verhoeven, Aarts and Swinkels, 1995; Okada, Jaji and Fukushima, 1998). 2-opt has also been successfully used in other combinatorial optimisation problems, such as graph partitioning (Kernighan and Lin, 1970) and bus vehicle scheduling (Wren, 1972; Smith and Wren, 1981; Kwan and Rahin, 1999). Amongst the successful applications, vehicle scheduling is in the area whose problem characteristics are closest to driver scheduling. Driver

scheduling is generally more complex than vehicle scheduling. The common characteristics and the differences are first summarised in this section, then a successful application of 2-opt in vehicle scheduling is reviewed. The last part of this section will discuss the potential applicability of 2-opt for driver scheduling.

3.3.1 Vehicle scheduling versus driver scheduling

Both vehicle scheduling and driver scheduling are concerned with the allocation of a set of resources to timetabled activities in the public transport field.

In public bus and train transport, vehicle scheduling is to allocate a vehicle to each journey in such a way that the total number of vehicles required operating all the journeys and the total vehicle cost are minimised. Each journey is a timetabled activity that starts and ends at specified times and locations. Vehicle scheduling involves making feasible and economic links amongst these activities. Similarly, driver scheduling is to allocate drivers to each vehicle in operation in such a way that the number of drivers and/or total cost are minimised. The pieces or spells of vehicle work are also timetabled activities that start and end at specified times and locations (i.e. ROs).

In vehicle scheduling, the feasibility of a solution depends mainly on whether there is enough time for the vehicle to connect between two successive journeys. However, in driver scheduling, the feasibility of a solution depends on not only the connection time between two successive vehicles to be operated by a driver, but also a set of complex labour agreement rules that governs the legality of the driver duties.

3.3.2 A bus vehicle scheduling system – BOOST

Applying 2-opt heuristics, a bus vehicle scheduling system VAMPIRES (Wren, 1972; Smith and Wren, 1981) was developed using the traditional procedural approach and later was superseded by BOOST (Kwan and Rahin, 1999) embracing the object oriented paradigm. BOOST first forms a crude initial schedule using a target number of buses. The target is either estimated by an automatic process or specified by the user. It is possible that some of the links in the initial schedule are time infeasible, i.e. the bus arrives later than it is due to depart. The schedule is then iteratively refined. In each iteration, a selected pair of links is broken up and re-linked in the alternative way if the overall infeasibility and/or cost of the schedule are reduced. At the end, BOOST returns either the lowest cost feasible solution it has found, or, if the target number of buses is too low, pointers to critical bus journeys that might be retimed to make the target bus number feasible.

BOOST has been tested using data from various bus companies and used in some feasibility studies involving bus companies in the UK and Brazil. Wren and Kwan (1999) report an application for one UK bus company, whose main depot in the city centre has been relocated to a new site. They have satisfactorily used BOOST to re-schedule their entire new operation.

3.3.3 2-opt heuristics for driver scheduling

Corresponding to journeys in vehicle scheduling, *spells* in driver scheduling are timetabled activities and can be linked to form an initial schedule. Similar to BOOST, the initial schedule could be refined iteratively by breaking and re-linking pairs of links. It seems that the problem could be solved by this method. However, further contemplation of the method has revealed the following difficulties.

- Costing method

Using Neighbourhood Search, a large number of solutions have to be evaluated during the scheduling process. The costing methods for vehicle and driver scheduling are different.

In vehicle scheduling, a link has an associated cost, which may be expressed as a function of *idle time* (i.e. waiting time of a vehicle after finishing a journey at a place and before making another journey from the place), *dead run* (i.e. non-passenger-carrying journeys) or *depot return* (i.e. a time gap long enough to justify temporarily returning the vehicle to the depot before its next departure). The cost of operating the actual journeys in most practical situations may be regarded as constant, and therefore for the optimisation process the total cost of a vehicle schedule can be regarded as the sum of the cost of all the individual links. This costing method is called a *link-based costing* method, in which each link in a vehicle (a chain of links) has an independent cost and the sum of the individual costs constitutes the cost of the chain. A revised chain can be easily costed by re-costing only the revised or newly-added links involving only simple addition and subtraction. For instance, in BOOST the cost difference between a current schedule and the new schedule after a 2-opt exchange between two links is just the same as the cost difference between the two pairs of links before and after the exchange.

However, in driver scheduling, it is generally not easy to assess precisely how much a link is contributing to optimality, and hence the cost of a duty. For example, all duties under eight hours might be paid the same. A simplification could be to treat all the links in the same duty as having the same cost as the whole duty. When any link in a duty is

changed, the duty has been transformed, the cost of which must be re-calculated according to its constituent links and the labour agreement rules. This costing method is called a *chain-based costing* method, which is more complicated and time-consuming than the link-based costing method used in vehicle scheduling.

Costing a duty (i.e. a chain of links) is complex. There are usually different types of duty (see duty types in Section 1.1.4), and each type has a corresponding set of rules governing the legality and payment of the duties. For example, the cost of a duty can be defined as the spreadover, or only the driving time, or the driving time plus partial unproductive working time.

- Penalty costing

In BOOST, a link is infeasible only when there is not enough time to connect two consecutive journeys. The quantitative penalty of a link is simply defined as the time lacking for connection, and thus the total penalty of a vehicle is defined as the sum of the penalties of all the individual links. The penalty of a feasible link is defined as zero.

However, in driver scheduling, it cannot be determined whether an individual link is feasible without considering the whole duty. Hence, the chain-based costing method is applied to treat all the links in the same duty as having the same penalty as the whole duty. If a duty is legal according to labour agreement rules, the penalty of the duty is defined as zero; otherwise, a quantitative penalty should be assigned to the duty. To add penalty costs to infeasible duties, the infeasibility of duties must be classified according to labour agreement rules, and then each type of infeasibility is assigned a corresponding penalty function. The total penalty of a duty can be defined as the sum of

all the individual penalty functions. Penalty costing a duty is non-trivial (see examples in Section 3.4 and Section 4.2.3).

- Terminal ends

  In vehicle scheduling, journeys are continuous activities of fixed timing. Each journey starts and ends at fixed time. Given an initial schedule, an optimal schedule could be generated if an optimal set of links connecting all the journeys could be obtained by breaking the links and re-linking the journeys. However, in driver scheduling, the lengths of driving activities (i.e. spells) are determined by the optimisation process subject to the availability of relief opportunities. Given an initial schedule, the set of spells is fixed. Starting from a set of crudely determined spells, an optimal schedule could not be guaranteed even though an optimal combination of the spells could be obtained by breaking and re-building spell-links.

  An alternative way is to use *pieces of vehicle work* (pieces for short) instead of spells as timetabled activities and to form a driver schedule by linking all the pieces. A duty can therefore be presented as a sequence of *piece-links*. Each piece-link connects two consecutive pieces of work and each piece starts and ends at fixed times. However, the number of links would be greatly increased, and the evaluation of duties and links becomes more complex and time-consuming. Spells have to be determined whenever a duty is formed.

To conclude, the BOOST system is a basis for the initial development of the 2-opt heuristic approach for driver scheduling, but extensive adaptation is expected. The driver scheduling problem is different from the vehicle scheduling problem and is generally more complex.

3.4 Simplified driver scheduling problem

The driver scheduling problem is a hard combinatorial problem. Compiling an effective driver schedule is very complex and difficult since there are many rules governing the legality of duties and the partition of blocks of vehicle work is restricted by the availability of relief opportunities.

To make the driver scheduling problem simpler to solve, as a starting point, we simplify the problem by only allowing the compilation of duties with up to two-spells according to a reduced set of labour agreement rules.

- Compile only the duties containing up to two-spells

The driver scheduling problem, which allows duties with three or more spells is much more complicated than only allowing duties with up to two-spells. In this initial research, we limit the number of spells in a duty up to two. This simplification has practical significance and was applied in most of the early heuristic driver scheduling approaches and in the Cavique et al's work reviewed in Section 2.4.1. The reason may be that two-spell duties are practically preferable, especially in bus operations. Moreover, some organisations only require duties with one or two spells, e.g. the Lisbon Underground situation reported by Cavique et al (1999).

- Enforce only some essential labour agreement rules on duties

The common labour agreement rules for both bus and train drivers have been outlined in Section 1.1.5. These rules governing the legality of a driver duty affect profoundly the complexity of duty compilation. For constructive approaches, the more rules that are imposed

on the validity of a duty, the more difficult it is to construct a solution. Hence, in this initial research, we simplify the problem by enforcing only the following rules, which would meet the essential requirements of most bus operators in the UK.

1) The total spreadover of a duty must be within a specified range.
2) The total daily working time of a duty must be limited to a certain number of hours.
3) The break between the two spells of work in a straight duty must be long enough for a driver to have a meal and to travel to start the next piece of work, while the break in the middle of a split duty must be longer than a certain amount of time.
4) There is a maximum length for a spell with or without a stop, and a maximum length for a stretch.
5) For split duties, the starting time must fall within a specified period, and they must finish before a specified time. They also have an earliest sign on and a latest sign off time.

If a duty obeys all the above rules, it is called a *valid duty* or *a feasible duty*; otherwise an *invalid duty* or an *infeasible duty*.

- Duty types

As discussed in Section 1.1.4, duties can be classified into two principal types: *split duties* and *straight duties*. Straight duties can be further divided into different types, e.g. *early duties*, *middle duties*, *late duties*. Some straight duties contain mealbreaks while the others do not. In this initial research, we classify duties into three types: *split duties* (having a long spreadover with a long break in the middle), *straight duties* (having a mealbreak in the middle), and *non-mealbreak duties* (i.e. straight duties with no mealbreak). Since only the duties with up to two-spells are allowed, both split duties and straight duties contain two spells with a mealbreak in

between, while the non-mealbreak duties may contain either one spell or two spells with a join-up in between. This classification can meet the requirement of many operators in the UK.

- Cost function and penalty function

Every duty has a corresponding payment, which is equal to the working time of the duty, and is called a *wage cost* or simply *cost*. Whether and how much non-driving work is counted as part of drivers' working time depends on operators, and the way to calculate the working time varies dependent on the type of the duty. The following gives an example of the cost functions:

Wage cost of a straight duty = spreadover

Wage cost of a split duty = sign-on-at-depot + start spell + end spell + sign-off-at-depot + sign-off before the long break + sign-on after the long break

Wage cost of a non-mealbreak duty = spreadover

Some operators stipulate a minimum payment for duties. If the counted working time of a duty is less than the specified minimum payment, the cost of the duty is assigned the minimum payment.

Usually, drivers prefer straight duties to split duties and operators do not prefer non-mealbreak duties. We treat the preferences as soft rules, which are catered for by means of some weighted costs to be minimised. For example, in the optimisation process, the cost of a split duty can be assigned twice its wage cost, the cost of a non-mealbreak duty can be assigned five times its wage cost, while the cost of a straight duty is assigned its wage cost.

If a duty is invalid, a quantitative penalty should be calculated to reflect how much the duty violates the labour agreement rules. Each rule has a corresponding kind of penalty, which is

defined as the difference between the actual time allowance and the due limit. If the difference is negative, it means this kind of rule is satisfied and the penalty is defined as zero. The following provides the formulae for calculating the penalty of a duty.

The penalties for an invalid straight duty:

Penalty-max-spreadover = spreadover – maximum spreadover

Penalty-working-time = working time – maximum working time

Penalty-mealbreak = minimum mealbreak – mealbreak

Penalty-stretch = stretch – maximum stretch

The penalties for an invalid split duty:

Penalty-max-spreadover = spreadover – maximum spreadover

Penalty-min-spreadover = minimum spreadover - spreadover

Penalty-working-time = working time – maximum working time

Penalty-mealbreak = minimum mealbreak – mealbreak

Penalty-stretch = stretch – maximum stretch

Penalty-earliest-start-on = earliest start on time – start on time

Penalty-latest-start-on = start on time – latest start on time

Penalty-latest-finish-off = finish off time – latest finish off time

Penalty-earliest-sign-on = earliest sign on time – sign on time

Penalty-latest-sign-off = sign off time – latest sign off time

The penalties for an invalid non-mealbreak duty:

Penalty-max-spreadover = spreadover – maximum spreadover

Penalty-working-time = working time – maximum working time

The total penalty of a duty is defined as the sum of the individual penalties. Validity of spells is enforced all the time.

3.5 Outline of the 2-opt driver scheduling approach

The basic idea of 2-opt heuristics is simple but it is non-trivial to design an effective application of the approach to solve a combinatorial optimisation problem such as the driver scheduling problem. A series of intermediate solutions are created and destroyed. A good algorithm design is necessary to prevent it from endless iterations. Many alternative designs are possible. Some strategies for escaping from the local optima are also necessary. This section presents a model and outlines a solution method to the simplified driver scheduling problem based on 2-opt heuristics.

3.5.1 Modelling driver activities and duty

Blocks with relief opportunities provide the basis for forming driver activities. *Spells* (not pieces) are essential activities in a driver duty. *Links* that connect successive spells are artificial driver activities and are called *spell-links* (*links* for short when the concept is unambiguous). Since there are at most two spells in a duty, whenever a pair of spells is linked a duty is formed. Meanwhile, a *sign-on activity* is inserted before the start spell and a *sign-off activity* is added after an end spell. Only *spell* and *spell-link* activities are explicitly modelled as driver activities. The validity and type of a link is treated as the same as those of the duty it belongs to. Figure 3.1 shows a hierarchical classification of driver activities.

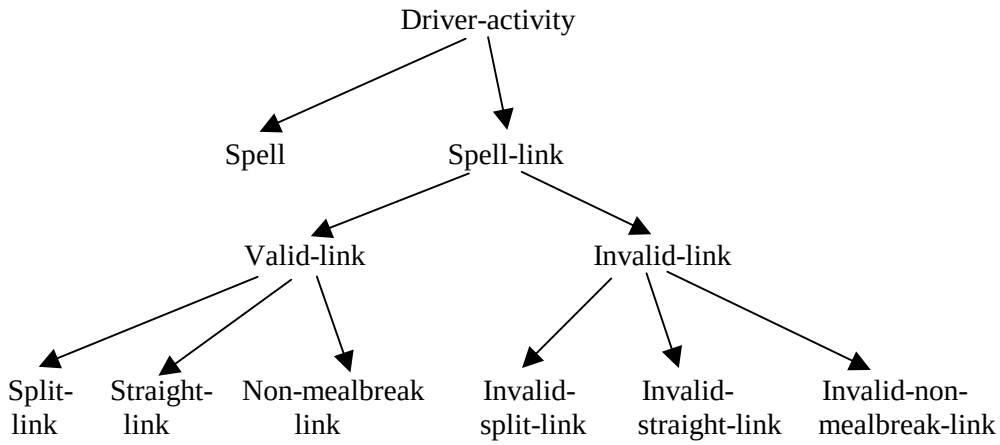


Figure 3.1: Hierarchy of driver activities

Each driver activity is associated with a *duty*. A *duty* consists of two spells (a *start-spell* and an *end-spell*) and a *link* in between (a *one-spell duty* is presented as a spell linked with a *dummy spell*, the work content of which is nil) while every spell has an implicitly embedded sign-on or sign-off activity. Figure 3.2 illustrates such a duty.

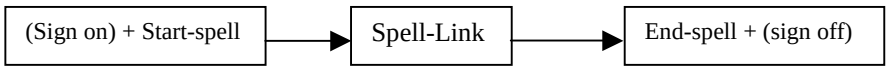


Figure 3.2: Illustration of a duty with two spells

3.5.2 General scheme of the 2-opt driver scheduling approach

An initial solution is the starting point of a neighbourhood search method. A good initial solution might be helpful, and therefore some heuristic approaches require a feasible initial solution. However, an algorithm that ensures a very good initial schedule would itself be complex and time-consuming. We use a very simple method to obtain a crude initial schedule. The initial schedule is required to cover all the vehicle work, but some duties may be invalid

because either there is not sufficient time for the driver to connect between two successive spells of work or the duty has violated some labour agreement rules. Then the initial schedule is iteratively improved in a 'mechanical' way. In the 2-opt heuristics, the refining process consists of two phases: *minimise-penalty* and *minimise-cost*. The phase of *minimise-penalty* is built on three major component processes: *swapping-link*, *replacing-ARO* and *adding-duty*. The phase of *minimise-cost* is built on the two processes: *swapping-link* and *replacing-ARO*. Each of these component processes consists of several operations, which together constitute multi-neighbourhood structures in the 2-opt driver scheduling approach. Figure 3.3 illustrates the general scheme of the approach. Further description is given in the following sections.

3.6 Formation of an initial schedule

A crude initial schedule is quickly constructed by the following steps.

a) Estimate a tight target number of duties N simply by the following formula:

$$N = V / (S - A - B - C)$$

where V denotes the total vehicle work. S denotes the maximum length of spreadover for a normal duty stipulated in the labour agreement rules. A and B denote the time allowances for drivers to sign on and sign off at depot respectively. C denotes the specified minimum meal break.

This formula guarantees that the target number is not higher than that achievable. The reason such a low target number is used is that the optimal number of duties could not be known in advance. The low target leaves room for the refining process to improve the solution by adding duties incrementally.

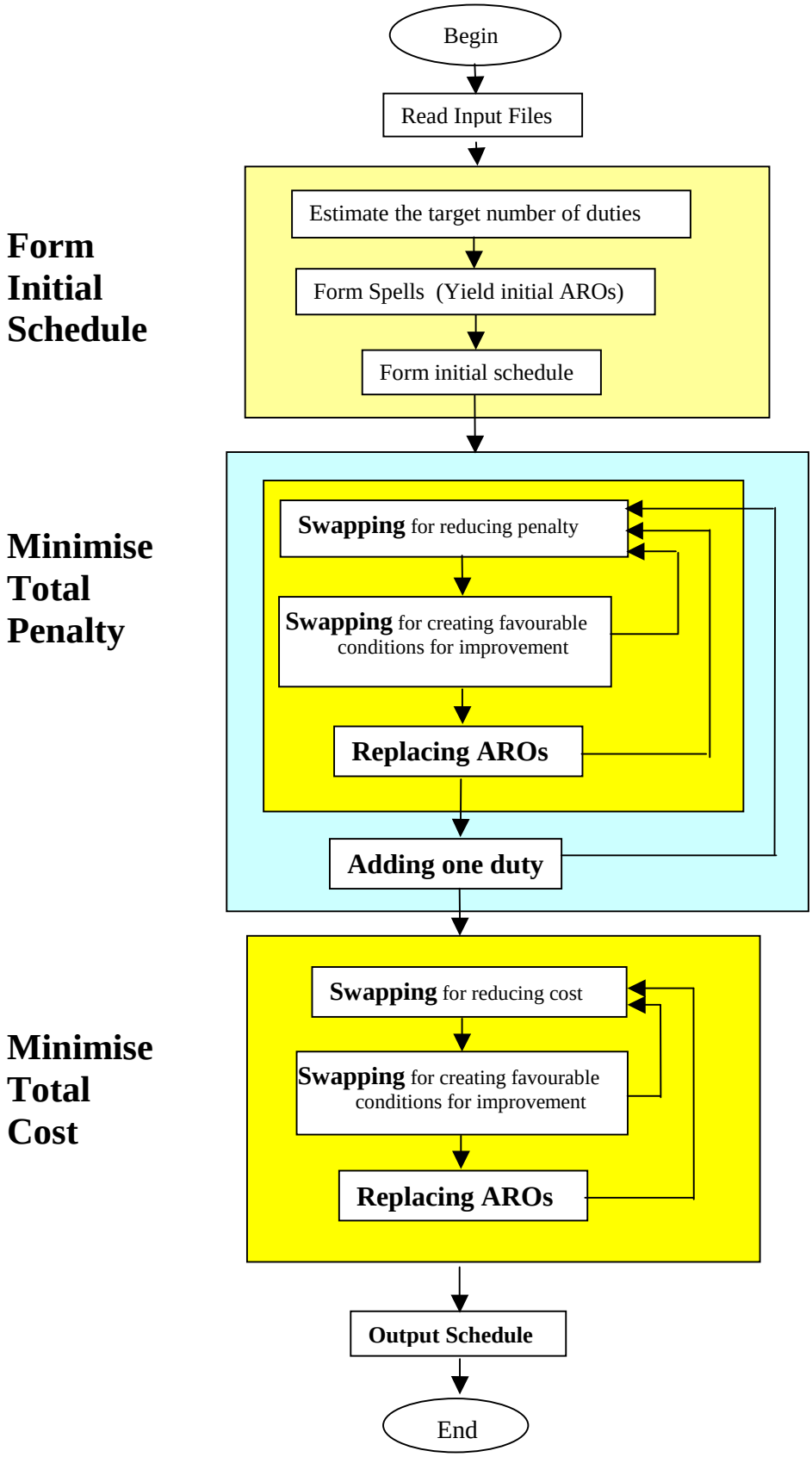


Figure 3.3: The major functional components of the 2-opt heuristics

b) Partition blocks into a set of spells.

The number of spells is defined to be twice the target number of duties since only duties with up to two-spells are considered. The average spell length can be calculated by dividing the total vehicle work by the number of spells. According to the average spell length, the blocks are partitioned into a set of spells without overlapping work. To ensure all the spells are valid according to labour agreement rules and together cover all the vehicle work, the number of spells required is allowed to be increased when necessary.

c) Sort the spells by starting time.

d) Form duties

Couple the spells in a simple manner to form driver duties such that the first spell has its starting time earlier than the starting time of the succeeding spell, and an arbitrary depot (the first depot in the data file) is assigned to each duty for the driver to sign on and sign off. Some duties may be time infeasible or violate some labour agreement rules.

If the number of spells is odd, generate the last duty by linking the remaining uncoupled spell to a *dummy spell*, the work content of which is nil. In this way, every duty can be treated as a two-spell duty.

When a duty is formed, a sign-on activity is inserted before the start spell, and an end spell is followed by a sign-off activity. Once a duty is formed, a spell-link between the two spells (including any dummy spell) of the duty is established. The spell-links are generated and deleted dynamically during the scheduling process.

The above method is very simple. An initial schedule can be quickly formed without considering much specific domain knowledge. An initial schedule can be constructed in many ways, such as those developed in the early heuristics. However, they are domain knowledge dependent and not readily portable to different transport operators, although the quality of the initial schedule may be better. The initial schedule constructed by the above method is very likely to contain infeasible duties. Therefore, in order to get rid of the infeasibility and to minimise the total cost of the schedule, a series of strategies is devised and included in the following functions: *swapping-link, replacing-ARO and adding-duty*.

3.7 Swapping links

The *swapping-link* process aims to improve the schedule by exchanging the links while the spells remain unchanged. 2-opt swapping-link to be presented in Section 3.7.1 plays a very important role in the refining process, which is driven by a set of *link-swapping rules* (described in Section 3.7.2). Link-swapping rules decide whether the selected links should be swapped during the scheduling process, and are classified into two groups: one is essential to improve the solution while the other is to help escape from local optima. Experiments on testing the swapping-link process are presented in Section 3.7.3. To improve the performance, some *k-opt swapping-link* processes have also been designed in Section 3.7.4.

3.7.1 2-opt swapping links

2-opt swapping-link constitutes the major part of the 2-opt heuristic approach, the principle of which is depicted as follows:

A schedule has a set of spells *S* and a set of spell-links *D*. A spell-link in *D* connects a pair of spells in *S*, which can be denoted as: $D = S \wedge S = \{ s_1 \wedge s_2 : s_1$ and s_2 are in $S\}$. The 2-opt heuristics select from *D* two distinct spell-links d_i and d_j, where $d_i = s_1 \wedge s_2$, $d_j = s_3 \wedge s_4$, s_1, s_2, s_3, s_4 in *S*. It then does a swapping trial by replacing the current two links with new links $d_i' = s_1 \wedge s_4$ and $d_j' = s_3 \wedge s_2$. A swap is allowed if some link-swapping rule is satisfied. Figure 3.4 illustrates the operation, where the depot for signing-on and signing-off in a duty must be the same. After swapping the links, two new duties are formed. The best depot for the driver to sign on and sign off is determined if more than one depot is used.

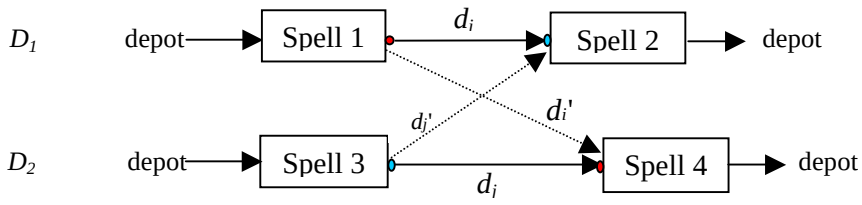


Figure 3.4: Illustration of 2-opt swapping-link (standard)

The above illustrates a typical scenario in applying the 2-opt method. However, in this way a start-spell can never become an end-spell while an end-spell can never become a start-spell. To ensure that any possible swapping scenarios are catered for, the 2-opt method is extended as illustrated in Figures 3.5-3.7. After swapping links, new best depots are determined for the new duties. The sign-on and sign-off activities are re-computed for the start-spell and end-spell respectively.

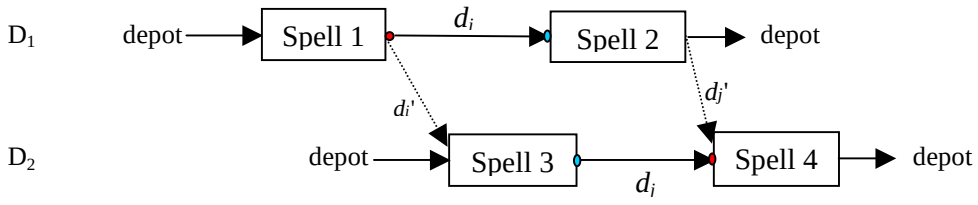


Figure 3.5: Illustration of 2-opt swapping-link (a)

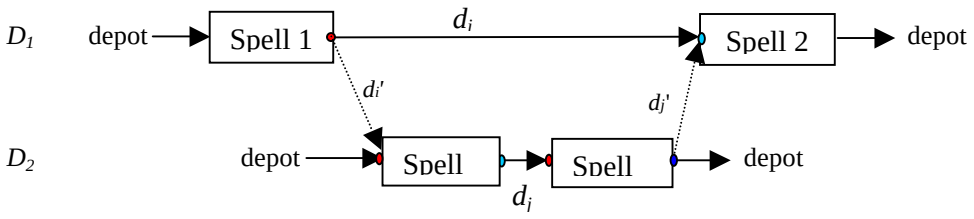


Figure 3.6: Illustration of 2-opt swapping-link (b)

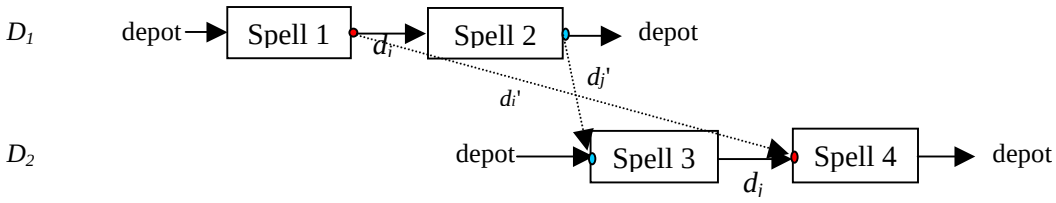


Figure 3.7: Illustration of 2-opt swapping-link (c)

In an initial schedule, every duty contains two spells, except one may contain only one spell with a dummy spell. Spells remain the same during the swapping process but may be reformed by a process to be discussed in Section 3.8, which can give rise to more single-spell duties. After swapping between two single-spell duties, a duty may result in having two dummy spells. This means the number of duties may be reduced during the swapping process.

3.7.2 Link swapping rules

Link-swapping rules are developed to decide whether the selected links should be swapped during the scheduling process. The rules can be divided into two groups, one of which is for improving the current schedule; the other is for creating favourable conditions for improvement, which helps escape from the local optima.

3.7.2.1 Swapping for improving the current schedule

Swapping-link is applied in both refining phases: minimise-penalty and minimise-cost for reducing penalty or cost respectively.

In the phase of minimise-penalty, swapping trials are carried out between two links, one of which is invalid. The links are swapped if the total penalty is reduced, and amongst the four ways of swapping-link the one that reduces total penalty most is chosen. Similarly, in the phase of minimise-cost, a swap is allowed if it reduces total cost and is the best among the four ways of swapping-link.

Obviously the best solution obtained, when the swapping-link process cannot lead to any improvement, is only a local optimum, which may not be the global optimum. Special swapping-link strategies are described in the next section, which may help escape from local optima.

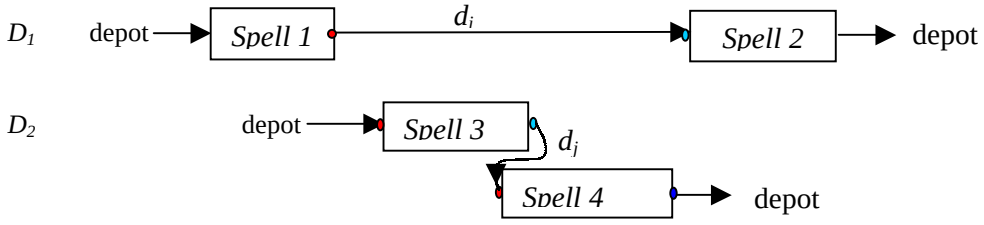
3.7.2.2 Swapping for creating favourable conditions for improvement

When no improving solution can be found, the rules called *wideBR/EvenBR, same-cost-wideBR/same-cost-EvenBR* and *wideSO/EvenSO, same-cost-wideSO/same-cost-EvenSO* are used to help escape from local optima.

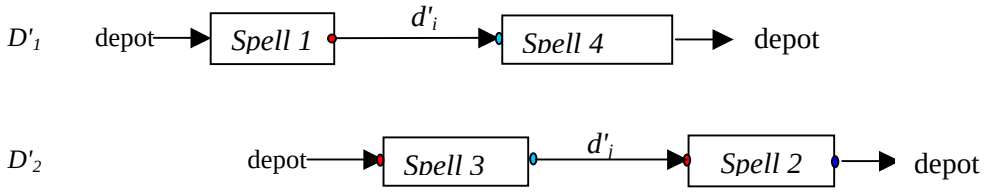
WideBR and *EventBR* allow a swap between any two valid links if the new generated links are still valid and have respectively a less even or a more even distribution of link lengths compared to before the swap. The WideBR and EvenBR swapping criteria, instead of

decreasing the total penalty, are intended to create more opportunities for reducing penalty later. We shall explain this by reference to Figure 3.8.

After the swap of widening break, we may form a duty D_1 with a large gap between *spell 1* and *spell 2*. Suppose that there is an invalid duty D_2 consisting of two spells: *spell 3* and *spell 4*. The swapping for reducing penalty operation may form the two new duties D'_1 and D'_2.



(a)



(b)

Figure 3.8: Illustration of the possible effect of the WideBR swap

The EvenBR rule may help in forming preferable links for drivers. Both WidenBR and EvenBR can also disturb the current schedule and further make the swapping for reducing penalty operation work.

Similarly, *same-cost-WideBR* and *same-cost-EvenBR* are intended to create more opportunities for reducing total cost. They allow a swap between two valid links if the new links are valid, cost the same or less, and have respectively a less or more even distribution of link lengths.

Having the same purpose as WideBR and EvenBR, the rules *WideSO* and *EvenSO* allow a swap between any two valid links if the new generated links are valid and have respectively a less or more even distribution of spreadover compared to before the swap. WideSO makes some duties with longer spreadover while making the others with shorter spreadover. The duties with a long spreadover are usually regarded as good duties while the duties with a short spreadover have some chance of being merged or to improve other duties by swapping spell-links. Similarly, *same-cost-WideSO* and *same-cost-EvenSO* are intended for assisting in reducing total cost by adjusting the distribution of spreadover of any two selected duties without increasing the total cost.

Other rules such as widening/narrowing the gap of the length of driving work or mealbreak can be applied in a similar way. The more rules that are applied, the more opportunities it has to improve the solution, however, the more computation is spent.

3.7.3 Experiments on 2-opt swapping

The 2-opt heuristic approach for driver scheduling has been implemented and tested. Two data sets were used at this stage. One problem was from the former Greater Manchester Buses (GMB), which contained 21 blocks and had a total bus working time of 227 hours and 52 minutes. The other was from Merseyside Transport (MTLD), which had a total working time of 508 hours and 35 minutes undertaken by 37 buses. The TRACS II system (in the latest version at the time of the experiments) was used for benchmarking. Throughout this chapter, TRACS II was run not allowing duties with three and four spells to be generated.

During the 2-opt swapping process, the spells never change. Therefore, the experiments were designed as follows. First, take the spells in the TRACS II solutions as the set of spells for forming an initial schedule; then apply the 2-opt swapping method to refine the initial schedule. The expectation is to obtain schedules as good as the TRACS II solutions, which were believed to be the best solutions to the test problems. Unfortunately, the results for both problems were not, although comparable to, as good as expected. More powerful swapping functions were needed, which led to the 3-opt, 4-opt, and even 5-opt swapping functions designed in the next section.

During the experiments, it was also found that the swapping to create favourable conditions was very helpful in improving the solution in the first few iterations, but thereafter it became ineffective. Another finding was that when several types of swapping-rule for creating favourable conditions were used, only the first one or two could profit. Moreover, each additional swapping rule added increased the computational time considerably. However, later experiments showed that the application of some of the rules was critical to some problems. We cannot simply omit one of them. A compromise was made by ensuring flexible selection of the rules for creating favourable conditions. Users can choose the rules to be used, the rules can also be chosen automatically according to the size of problems. For example, when the estimated target number of duties is less than 50, all the link-swapping rules are used, otherwise only two rules WideBR/EvenBR and WideSO/EvenSO are used to create swapping opportunities.

3.7.4 K-opt swapping links

To enhance performance, *k-opt* (k = 3, 4, or 5) methods are designed, which do swapping trials among k links. They are illustrated in Figures 3.9-3.11, where a two-direction arrow denotes a

link with two possibilities, for example in the duty D_1, the link could be either the spell *s1* followed by the spell *s2* or *s2* followed by *s1* depending on the starting times of the spells. The spell that starts the work earlier becomes the start-spell. The dotted arrows denote the new links to be generated.

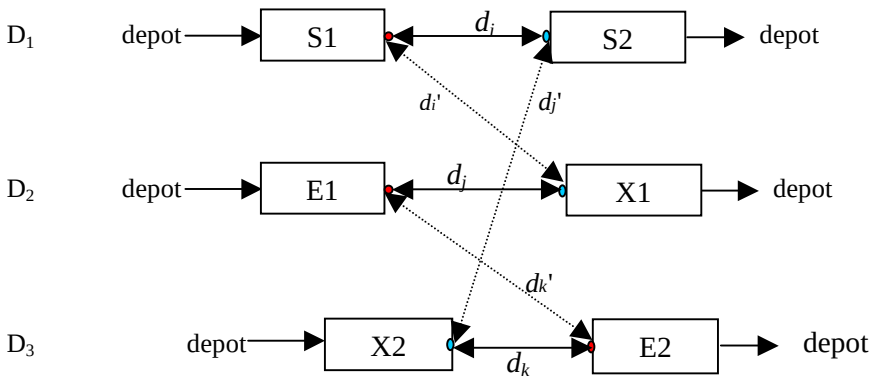


Figure 3.9: Illustration of 3-opt swapping-link

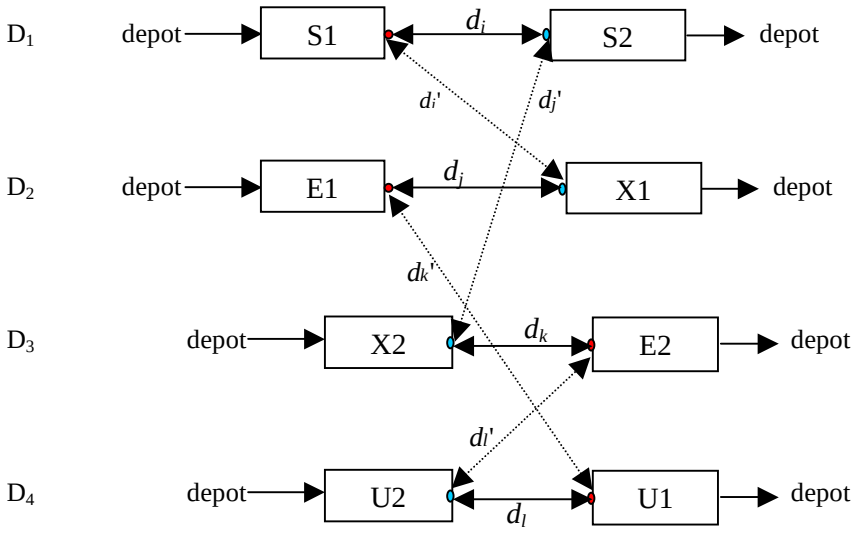


Figure 3.10: Illustration of 4-opt swapping-link

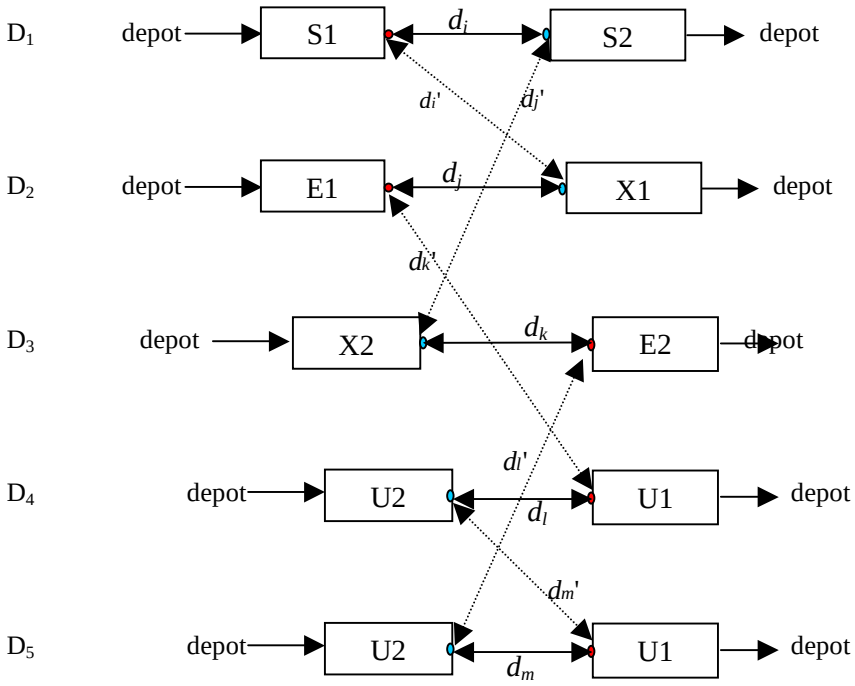


Figure 3.11: Illustration of 5-opt swapping-link

Applying the k-opt swapping functions, the experiments described in the previous section were carried out again and the same results as the TRACS II solutions were obtained. It could be concluded that near-optimal solutions would be achievable given a good set of spells (or AROs) with intensive swapping algorithms.

A good swapping process is very important. However, it would be extremely hard, if at all possible, to determine a good set of spells from raw data. Working from a crudely derived set of spells, an optimal solution to the problem could not be guaranteed even if an optimal combination of the spells could be obtained by swapping operations. Considering the whole scheduling process, it is more crucial to attain high computational efficiency. The 3-opt swapping method is therefore only used to reduce total penalty, and is optional because it takes a long computational time. The 4-opt and 5-opt swapping methods took a considerably long

time and were only tested for very small problems in which the numbers of duties was less than 15. Due to the low computational efficiency, the 4-opt and 5-opt swapping-link processes were abandoned.

3.7.5 Summary

Each swap in the 2-opt or k-opt methods causes two or *k* links to be removed and replaced by newly generated links while the spells remain the same. Once a link is generated, a new duty is formed and the best depot is assigned. During the swapping process, it is possible to reduce the total number of duties. The whole swapping process terminates when no more improvement can be made.

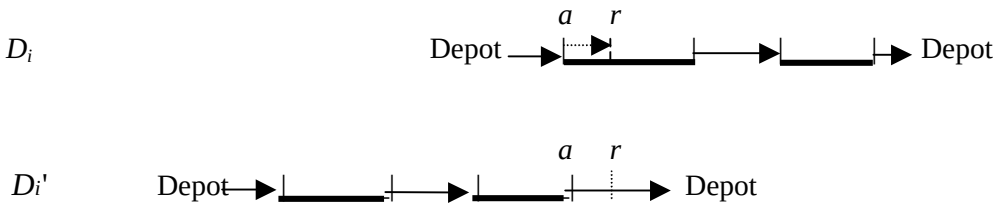
3.8 Replacing AROs

As already mentioned, during the swapping-link process the spells are not changed, i.e. the AROs are fixed. The selection of AROs depends on the first block cutting, which may be very rough. It is therefore necessary to revise the selection of AROs, which leads to the function *replacing-ARO*. *Replacing-ARO* attempts to use alternative relief opportunities to replace some current AROs in order to reduce total penalty or total cost.

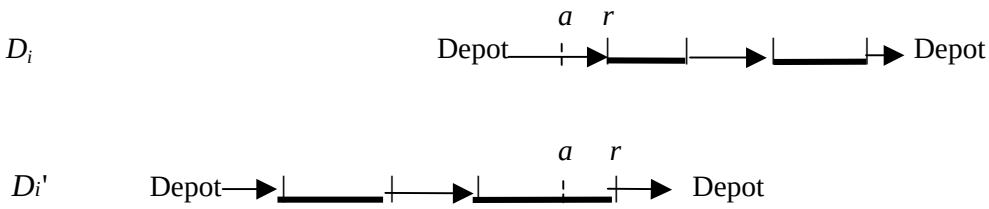
3.8.1 Principle of replacing AROs

The principle of replacing-ARO is explained by reference to the example illustrated in Figure 3.12.

We consider replacing the ARO *a* in the duty D_i in Figure 3.12(a). Suppose that *r* is the nearest next relief opportunity to *a*, and the replacement of *a* with *r* can reduce the penalty or the cost of D_i. After the replacement, the piece of work between *a* and *r* is left unassigned to any driver. To solve this problem, another duty D_i' also using *a*, called the *relevant duty*, must be found and extended to cover the unassigned piece of work. Figure 4.12(b) shows the new duties after the replacement. Whenever a new duty is generated, a best depot is selected accordingly.



(a)



(b)

Figure 3.12: Illustration of replacing AROs

Note that replacing-ARO might reduce the number of spells. Suppose the start-spell in D_i in Figure 4.12(a) contains only one piece of work, i.e. *r* is another ARO. After the operation, the entire work between *a* and *r* is moved to the duty D_i' and the duty D_i only contains the end-spell. A *one-spell duty* is therefore generated while the start-spell becomes a dummy spell. If

replacing-ARO is operated on a non-dummy spell in a one-spell duty that contains only one piece of work, the duty will be removed after the operation.

3.8.2 Strategies of replacing AROs

A series of strategies are designed to replace AROs efficiently. In the phase of minimise-cost, each ARO is considered for replacing ARO. A replacement is allowed when total cost is reduced. However, in the phase of minimise-penalty, attempting replacement of each ARO is not efficient. An obvious reason is that the replacement of an ARO related to two valid duties would not reduce any infeasibility. Hence we stipulate that the replacement of AROs is limited to invalid links and their relevant links. The efficiency of replacing AROs to reduce infeasibility is also affected by how the replacement is conducted. Before addressing this, we need to identify the types of infeasibility, which are listed below.

1. *Invalid Spreadover*, the spreadover is out of the range of spreadover.
2. *Invalid Sign-on*, the sign-on is earlier or later than the time stipulated.
3. *Invalid Sign-off*, the sign-off is earlier or later than the time stipulated.
4. *Invalid Start-Stretch*, the start-stretch is longer than the limit.
5. *Invalid End-Stretch*, the end-stretch is longer than the limit.
6. *Invalid Mealbreak*, for a two-spell straight link, there is insufficient time for the driver to have a meal break and then to make the connection between two successive spells of work; for a split duty, the gap between the two spells is less than the minimum long mealbreak stipulated.
7. *Invalid Working-time*, the total counted working time exceeds the maximum daily working time stipulated.

We define a *relevant ARO* to be an ARO whose replacement may reduce infeasibility. Different types of infeasibility correspond to different relevant AROs, which are represented by black circles in Figure 3.13. Other unclassified infeasibility is grouped as *others*, where all the AROs in a duty are regarded as relevant AROs.

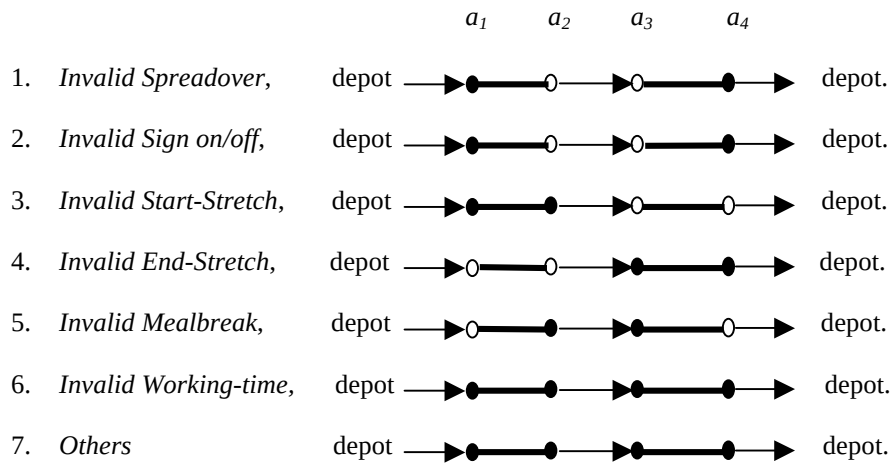


Figure 3.13: Types of invalid duties and the relevant AROs for replacement

For each invalid link, only the corresponding *relevant AROs* are considered in the replacing-ARO operation in order to save computational time. It is suggested that the operation is carried out on all the invalid links in the order as listed in Figure 3.13 rather than randomly, i.e., work first on all the links with invalid spreadover, then on those with invalid sign on/off, and so on. This order is based on the following reasons.

Limit on the spreadover of a duty is related to the duty's type. Extending or shortening the spreadover of an invalid duty might make its type change so that it would become valid according to the rules associated with the new type. Moreover, it is hoped that an invalid duty could be improved or made valid by replacing as few AROs as possible. Replacing-ARO operation is therefore done first on the duties with spreadover infeasibility, and the last two

types of infeasibility are assigned a low priority because they consist of more relevant AROs. Below is the replacing-ARO process depicted by an example for the first case of where invalid spreadover is first considered.

Take a duty and check if the spreadover is invalid. If not, take another one; otherwise do replacing-ARO trials as follows by reference to Figure 3.13.

- First, consider a_1 if it is not the first relief opportunity in the block. Replace a_1 with the next earlier relief opportunity and find the relevant duty and let it cover the left piece of work. Two new duties are temporarily constructed.
- Then, consider a_4 similarly if it is not the last relief opportunity in the block.

The replacement that leads to greater improvement is selected. If none of them refines the schedule, take another duty until all the duties have been tried.

In the 2-opt heuristic approach, every link with its relevant links, if they exist, is considered one by one to do the replacing-ARO operation.

The replacing-ARO method could be extended to apply recursively the operation on the relevant duty if it is made invalid by the operation. However, a duty usually has four AROs, each of which may have a corresponding relevant duty, making the alternative method complicated.

Figure 3.14 provides an illustration of performing replacing-ARO on three links, where three duties $D_{i,}$ $D_{j,}$ $D_{k,}$ are involved. A thick line denotes an original spell while a thick dotted line denotes a new spell generated after replacing AROs. Such a replacing-ARO operation is allowed if the total penalty or the total cost is reduced.

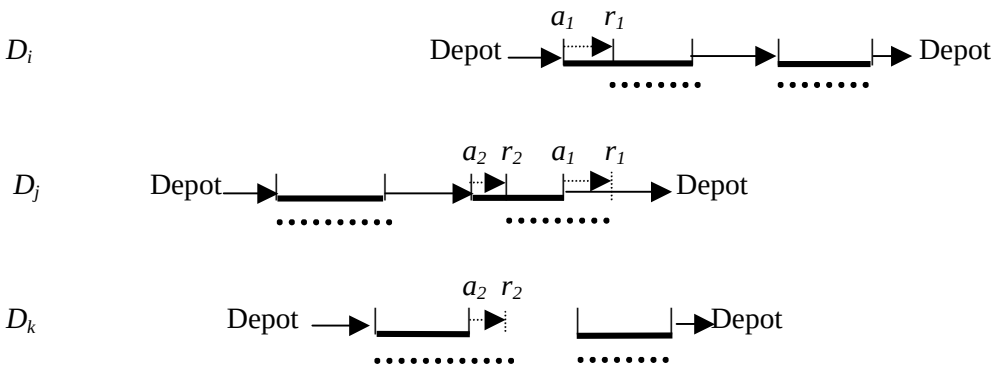


Figure 3.14: Illustration of replacing AROs on three links

To develop alternative strategies for replacing ARO requires a great deal of research, and it is beyond the purpose of this initial research, which aims to investigate the potential of the 2-opt heuristics to driver scheduling and to gain some experience for the future research.

3.8.3 Hybrid of swapping links and replacing AROs

In the 2-opt driver scheduling approach, replacing-ARO is iteratively applied to improve the solution after the swapping process cannot lead to any improvement. When replacing-ARO fails to improve the solution further, the swapping process is recalled. The swapping and replacing-ARO operations are employed in turn until none of them makes any further improvement.

Each swapping-link and replacing-ARO process is capable of refining the solution and therefore has an impact on the final solution. However, how an individual operation affects the final solution cannot be predicted. In the 2-opt heuristics, several neighbourhood structures such as those illustrated in Figures 3.4-3.7, 3.9-3.12, and 3.14 are applied. It is impossible to prove that any particular scheme of organising them would be optimal or better than another one. We

can choose to execute swapping-link and replacing-ARO in turn as is done currently. We could also choose other schemes such as a hybrid of swapping-link and replacing-ARO described below.

Take a pair of links and do swapping trials first. If the swapping trial is not successful, try replacing AROs on each link. If replacing-ARO is successful, do swapping-link trials again. This can be called a *compound swapping operation* incorporating replacing-ARO operation.

Obviously it is very hard to consider all possible algorithmic design combinations. The above discussion has highlighted the difficulty of designing a structure or a strategy in the 2-opt heuristics.

3.9 Adding duties

So far, the number of duties used in the series of refining schedules does not exceed the target number, which is set deliberately low and may be lower than achievable. The best schedule obtained may still be infeasible. To obtain a feasible solution, more duties may need to be introduced. A process called *adding-duty* is devised in which,

- One duty is added at a time when the swapping-link and replacing-ARO processes cannot lead to any improvement;
- Some work from the current duties is taken out to form the extra duty, such that the overall total penalty is reduced.

Two ways of adding a duty are developed: *two-link-adding* and *one-link-adding*. Two-link-adding handles two invalid duties at a time. If two-link-adding fails, one-link-adding is applied, which deals with one invalid link at a time. Figure 3.15 shows an example of two-link-adding.

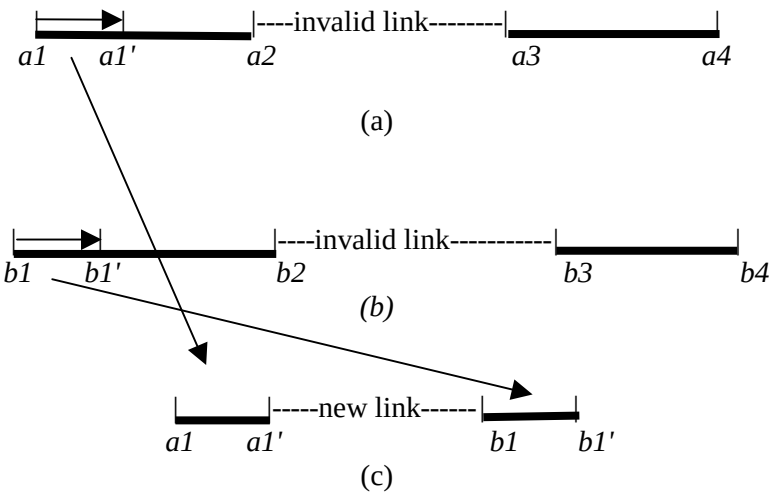


Figure 3.15: Example of two-link-adding-duty

Figure 3.15 shows two invalid duties (a) and (b). Suppose both of them are spreadover infeasible and each duty may be shortened by removing one piece of work at the beginning or at the end. A new duty (c) is allowed to be added to cover the removed work if total penalty is reduced.

One-link-adding is to improve an invalid duty in the following two ways. In the first way, one or more consecutive pieces of work are first cut from the invalid duty such that the infeasibility of the duty is removed or reduced, and then the cut work is linked with a piece of work cut from a valid duty such that the total penalty is reduced. The other way is to separate the invalid duty into two duties such that the total penalty is decreased.

As in the swapping-link and replacing-ARO processes, there may be other possible methods for adding duties. However, the method described above seems to be adequate.

3.10 Implementation and results

A computer program has been developed to implement the 2-opt heuristic approach using Borland C++ (Version 5.02). Many experiments and tests have been undertaken. Results for five of the test problems are presented below. For all the test problems, TRACS II was run by disallowing the generation of the duties with three and more spells. During such runs, some vehicle work remained uncovered in the duty generation process according to the labour agreement rules. The uncovered pieces of work were removed, and then the problems were solved by TRACS II and the 2-opt heuristic approach under the same conditions.

During a real-life problem solution process, TRACS II may encounter the situation, in which some pieces of work remain uncovered according to the labour rule parameters, amongst which some are filtering rules added by TRACS II to prevent the set of potential duties generated from being prohibitively large. It is not allowed to simply remove the uncovered pieces of work. The usual practice is to adjust the parameters to cover all the pieces of work. This may be unachievable or may be achieved by many adjustments. Another practice is to make a special run, after the previous runs, to generate duties covering specially the uncovered pieces of work based on a set of relaxed rule parameters. It may need several special runs to cover all pieces of work, and finally the duties generated by all the runs are merged. The 2-opt heuristic approach does not have such a problem; it produces a solution in one go.

3.10.1 Greater Manchester Buses

A driver scheduling problem from the former Greater Manchester Buses (GMB) was chosen as a test problem. This problem contained 21 blocks and had a total bus working time of 227 hours and 52 minutes. The original problem allowed three and more spell duties. After switching off

the generation of three and four spell duties, some work was uncovered. The modified GMB problem contained 15 buses into 16 blocks with 217 hours and 29 minutes of bus work per day. The total number of relief opportunities was 161. Table 3.4 summarises the solution and compares it with that obtained by TRACS II.

Table 3.4 Comparison between 2-opt and TRACS II solutions to the GMB problem

System	No. Duties	Cost (in hours)	Elapsed Run Time (hh:mm:ss)
TRACS II	32	273:34	More than 5 minutes
2-opt	34	273:10	0:00:39

A smaller problem called GMBmini was created, which utilised the labour agreement rules in GMB. The vehicle work was fictitious and contained fourteen pieces of work distributed in three blocks. The results are displayed in Table 3.5.

Table 3.5 Comparison between 2-opt and TRACS II solutions to the GMBmini problem

System	No. Duties	Cost (in hours)	Elapsed Run Time (hh:mm:ss)
TRACS II	4	24:58	More than 5 minutes
2-opt	4	24:58	< 0:00:01

3.10.2 Merseyside Transport

A problem called MTLD was from Merseyside Transport, which had a total working time of 508 hours and 35 minutes undertaken by 37 buses. Table 3.6 compares the 2-opt solution and the TRACS II solution to this problem.

Table 3.6 Comparison between 2-opt and TRACS II solutions to the MTLD problem

System	No. Duties	Cost (in hours)	Elapsed Run Time (hh:mm:ss)
TRACS II	77	605:10	More than 20 minutes
2-opt	81	615:05	0:05:10

A smaller problem called MTLDmini was derived from the MTLD problem by only retaining five blocks while removing the other blocks. The data had 70 hours and 24 minutes of vehicle work. The results are displayed in Table 3.7.

Table 3.7 Comparison between 2-opt and TRACS II solutions to the MTLDmini problem

System	No. Duties	Cost (in hours)	Elapsed Run Time (seconds)
TRACS II	13	88:04	More than 5 minutes
2-opt	13	91:39	0:00:40

3.10.3 Regional Railways North East

A problem RRNE came from the former Regional Railways North East, which operated a large railway network service covering a wide area in northern England. The problem contained 197 blocks and had a total working time of 1944 hours and 36 minutes. There were 1656 relief opportunities and 46 relief points, amongst which 20 were depots. This data could not be handled by TRACS II in one go and it was divided into four sub-problems when the scheduling work was first commissioned. To test the 2-opt approach, the four sub-problems were merged back to one big problem. Unfortunately, no solution was obtained after 20 hours of running and the program was terminated. It was decided to investigate whether the TRACS II solution could be improved instead. The decomposition of a large problem compromises solution optimality. There should be some room for improvement. A new experiment was therefore designed, in which the TRACS II solution, containing the solutions of the four sub-problems combined, was used as a starting solution. The computational results are shown in Table 3.8.

Table 3.8 Comparison between 2-opt and TRACS II solutions to the RRNE problem

System	No. Duties	Cost (in hours)	Elapsed Run Time (hh:mm:ss)
Sub-problem 1	96	782:18	
Sub-problem 2	111	910:01	
Sub-problem 3	47	328:21	
Sub-problem 4	76	581:35	
TRACS II (total)	330	2602:15	About a day
2-opt	N/A	N/A	
2-opt (based on the TRACS II solution)	329	2596:39	1:14:55

3.10.4 Summary

This section has presented five experiments carried out on a 233 mHz personal computer. The elapsed times recorded were actual clock times to obtain the solutions. TRACS II did not report on elapsed times but took considerably longer time in general. For example, for the smallest problem it spent more than 5 minutes and for the largest problem about a day. The TRACS II solutions presented above are the best after tuning the parameters, and different versions of the duty generation program (BUILD) specific to different test problems were used (the latest development in TRACS II is such that there is now only one version of BUILD for all problems). In contrast, the 2-opt solutions were obtained in just one go. The 2-opt solutions are generally not as good as the TRACS II solutions. For the very large problem, the 2-opt heuristic approach did not generate a solution based on the infeasible initial schedule it constructed, but improved the TRACS II solution benefiting from not having to decompose the problem.

3.11 Conclusions

A 2-opt heuristic approach has been presented in this chapter. Some of the basic operations in the swapping-link and replacing-ARO processes were tried in some very early heuristic

approaches (e.g. Weaver, 1968; Weaver and Wren, 1970 and 1972). This research has designed a more elaborate scheme including heuristics for helping to escape from local optima.

The 2-opt approach presented in this section is purely heuristic based and incorporates multi-neighbourhood structures. It applies the 2-opt heuristics to driver scheduling not relying on specific problem knowledge. It is much more generally applicable to problems from different operators compared with the early heuristic systems. Since no specific problem knowledge is required, it is applicable to different situations subject to the simplified requirements. Compared with the generate-and-select based driver scheduling systems, the 2-opt heuristic system is much easier to manipulate because there are no artificial rule parameters for the user to adjust; and the computation time is considerably shorter.

However, the 2-opt heuristic approach has significant limitations. The basic 2-opt heuristic does not have a specific means of escaping from local optima. Although the use of different link swapping rules is helpful, the swapping for creating favourable conditions is not very efficient. Designing and organising the search strategies is very difficult because the number of possible strategies and their combinations would be too many. Furthermore, it is extremely hard to rank the alternatives in fixing the framework of the 2-opt approach. The most significant limitation of this current 2-opt heuristic approach is that it cannot compile three or more part duties. When more-spell duties are considered, it would be significantly more complicated to expand the current approach.

Despite the limitations, the 2-opt heuristic approach has its advantages and has demonstrated the potential of a constructive heuristic approach for driver scheduling. The experience is greatly helpful to the further research. The following summarises the crucial factors, identified for a more general approach.

- Swapping links and replacing AROs are the most important processes contributing to solution improvement. The replacing-ARO process offers the constructive heuristic approach potential to handle windows of relief opportunities because the replacing-ARO process could possibly select any ROs in the windows of relief opportunities as AROs.
- The use of a deliberately low target number of duties together with a carefully controlled increment of extra duties is helpful in minimising the number of duties.
- The cost function and penalty function are established.
- Multi-neighbourhood structures are defined by swapping-link and replacing-ARO operations.

Chapter Four

Modelling the Driver Scheduling Problem and Overview of HACS

4.1 Introduction

After the research on the 2-opt driver scheduling approach presented in the previous chapter, research is carried further to solve the driver scheduling problem with the simplifications being removed: the number of spells in a duty is unlimited; the common labour agreement rules (listed in Section 1.1.5) are fully enforced on duties; more duty types are considered, and windows of relief opportunities are handled. Handling WROs and allowing an unlimited number of spells in a duty are currently beyond the capability of TRACS II.

The review of driver scheduling methods in Chapter two has found that the most successful existing driver scheduling systems are based on ILP, while some other generate-and-select

approaches have taken advantage of the LP relaxation solution. Inherent in the generate-and-select approach, artificial restrictions are usually imposed at various stages to control the size of the problem and search space. The early heuristic approaches solved the driver scheduling problem by simulating human schedulers' methods instead of building a general model. As discussed in Chapter two, a good constructive heuristic approach can be less dependent on domain knowledge and be free of artificial restrictions. This enables the building of a model, which better matches real-life problems, and hence has a chance of yielding better solutions.

The research presented in the previous chapter has demonstrated the viability and potential of the constructive heuristic approach. This chapter will first present two formal models for the driver scheduling problem (Shen and Kwan, 2000a and 2001a). Based on these models, a constructive driver scheduling approach using the Tabu Search framework is developed, which is overviewed at the end of this chapter.

4.2 A formal model for driver scheduling

There are four main aspects in modelling bus and train driver scheduling problems, which are *vehicle work*, *driver schedule*, *labour agreement rules*, and *objectives*.

4.2.1 Vehicle work

As described in Chapter one, the work in a vehicle schedule can be represented as a set of blocks. For driver scheduling purposes, each block is usually presented in a graphical format identifying a sequence of relief opportunities. However, the existence of windows, which provide a range of time instead of a particular time, for relief opportunities are usually ignored. The reason of simplifying WROs into ROs by the existing approaches is that WROs

dramatically increase the magnitude of problem to beyond the capability of the existing approaches. The following model for vehicle work incorporates WROs. The notations are extended from those in Chapter one.

$B = \{B_i \mid i=1,2,\ldots,k\}$	*a block set*, i.e. a set of all the k blocks in a problem.
$(B_i, \leq) = \{b_{i1} < b_{i2} < \ldots \leq b_{im_i}\}$	*a block*, defined by a linearly ordered set of WROs b_{ij} ordered by relief time, where m_i is the number of WROs in B_i.
$b_{ij} = (u_{ij}, w_{ij}, v_{ij})$	*a WRO*, defined by a ternary vector, where $u_{ij} = [t_1, t_2]$ denotes a time window from time t_1 to time t_2. w_{ij} is a relief point, $v_{ij} \in \{0,1\}$, $v_{ij}=0$ if b_{ij} is an U-WRO and $v_{ij}=1$ if b_{ij} is an A-WRO.

In the above definition of a WRO, the first subscript identifies the block, to which the WRO belongs. For example, b_{ij} is the j^{th} WRO in block B_i. The subscripts may be omitted when they are obviously unambiguous.

A *RO* is a special case of a WRO, where $t_1 = t_2$. Sometimes we define a *RO* by an ordered pair (t_{ij}, w_{ij}), where t_{ij} and w_{ij} are *relief time* and *relief point* respectively.

4.2.2 A driver schedule

A *driver schedule* is a solution of the driver scheduling problem, which contains a set of valid *driver duties* that together cover all the vehicle work in *B*. Under the generate-and-select approach, only valid potential duties are generated, and then various methods can be employed to find a subset to form a schedule. In a constructive approach, only allowing valid duties at all stages may be difficult and computationally expensive. In contrast, covering all the vehicle

work would be easy if validity of the duties is not a prerequisite. Also, allowing infeasible intermediate solutions may enable more flexible exploration of the solution search space. Therefore, we model a *schedule* simply as a set of duties that together cover all the vehicle work. Some duties may be illegal because either there is not enough time to make a connection between two successive spells, or the duty has violated some labour agreement rules.

A *duty* starts with a sign-on activity, followed by a sequence of spells, and ends with a sign-off activity. Each *spell* includes one or more consecutive *pieces of work* and at least two *ROs*. The first and the last ROs are AROs, each of which corresponds to a unique WRO in a certain block. Figure 4.1 illustrates a duty, where an arrow denotes a link between two driver activities. *Sign-on time* at *depot* and *sign-off time* at *depot* are treated similar to AROs when linkages are considered.

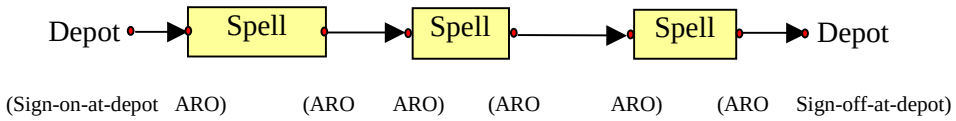


Figure 4.1: Illustration of a driver duty

In Figure 4.1, a *duty* can be defined by a sequence of *links* (called *spell-links*). Each *spell-link* consists of an ordered pair of AROs (the first is called an *arrival-ARO* while the second is called a *departure-ARO*), which connects two successive spells in the duty, the sign-on activity to the first spell, or the last spell to the sign-off activity. Formally, a solution S to a problem with a block set B is defined as follows: let D_i be a duty in S.

$p_{ij} = (t_p, w_p)$ *an arrival-ARO*, corresponding to a unique WRO $b_{lk} = (u_{lk}, w_{lk}, v_{lk}) \in B_l \in B$ such that $t_p \in u_{lk}$ and $w_p = w_{lk}$.

$q_{ij} = (t_q, w_q)$	*a departure-ARO*, corresponding to a unique WRO b_{lk} $= (u_{lk}, w_{lk}, v_{lk}) \in B_l \in B$ such that $t_q \in u_{lk}$ and $w_q = w_{lk}$.
$d_{ij} = (p_{ij}, q_{ij})$	*a spell-link*, defined by a directed pair of an arrival-ARO and a departure-ARO.
l_i	*the number of links* in D_i.
$(D_i, \preceq) = \{d_{i1} < d_{i2} < \ldots < d_{il_i}\}$	*a duty*, defined by a linearly ordered set of links in the order they occur in the duty (the links may not be in chronological order). There is an associated function for returning the next link of a given link: $next(d_{ij}) = d_{i,j+1}, \; j=1,2,\ldots,l_i-1.$
n	*the number of duties in the solution.*
$P = \{p_{ij} \mid i = 1,2,\ldots,n; \; j = 1,2,\ldots, l_i\}$	*an arrival-ARO set.*
$Q = \{q_{ij} \mid i = 1,2,\ldots,n; \; j = 1,2,\ldots, l_i\}$	*a departure-ARO set.*
$S = (P, Q, D)$	*a solution*, presented by a ternary vector, where $D = \bigcup_{i=1}^{n} D_i$.
$\Sigma = \{S_i \mid i = 1,2,\ldots,N\}$	*a solution space*, i.e. a set of all possible solutions, where N is generally an enormous unknown constant in practical problems.

The relationship between blocks and duties is illustrated in Figure 4.2, where 7901 and 7902 are two blocks while D_i is a duty containing four links denoted by solid arrows. The duty D_i covers three spells of vehicle work in the blocks 7901 and 7902, which are from 0609 to 1049 in 7902, from 1213 to 1530 in 7901, and from 1552 to 1825 in 7902. Hence there are six AROs: q_{i1} = (0609, G), p_{i2} = (1049, W), q_{i2} = (1212, W), p_{i3} = (1530, H), q_{i3} = (1552, H), and p_{i4} = (1825, G),

and two pairs p_{i1} = (*sign-on-time, depot*), q_{i4} = (*sign-off-time, depot*) associated with sign-on/off activities and treated similar to AROs.

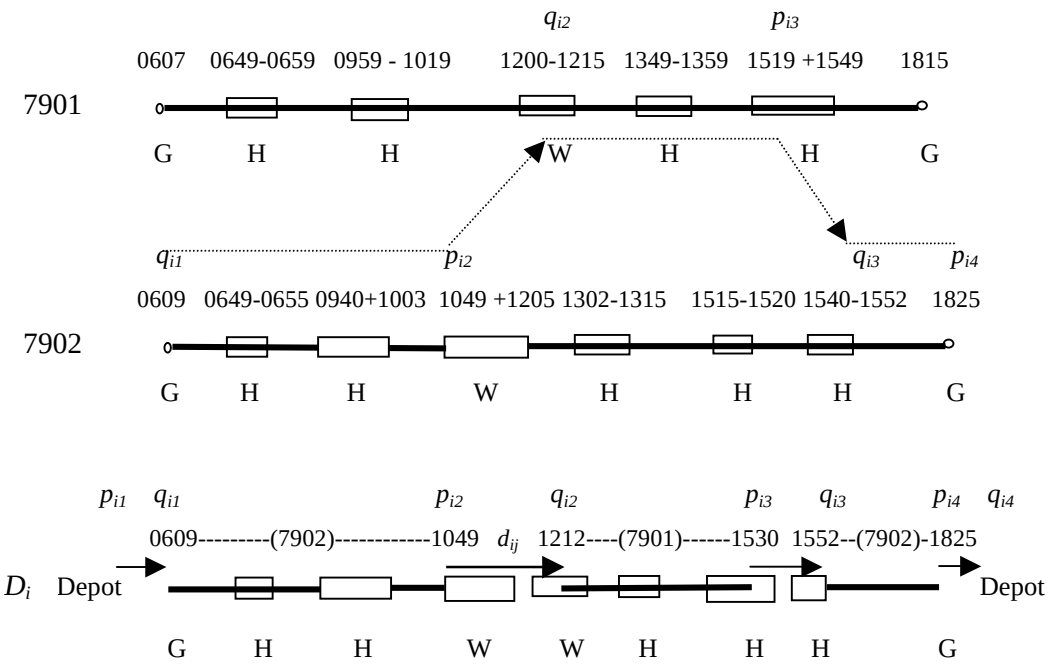


Figure 4.2: Relationship between blocks and duties

4.2.3 Labour agreement rules and costing functions

Labour agreement rules are operator-specific. A general discussion has already been given in Section 1.1.5. In this model, the following rules are enforced on duties:

1. The spreadover, the spells and the stretches fall within their respective limited ranges;
2. The duty signs on/off and starts on/finishes off within their respective limited time periods;
3. The duty signs on and signs off at the same depot;
4. The working time of the duty does not exceed a specified limit;

5. The mealbreak is not shorter than a specified minimum length of time, and the meal is taken within a restricted time range;
6. For a split duty, the longest gap between two consecutive spells is not shorter than a specified minimum length of time.

The labour agreement rules govern the legality of duties. There are different types of duty (see 4.2.3.1). Each type has an associated set of labour agreements. The legality and payment of a duty is governed by the set of rules associated with its type, and can be quantified by penalty and cost functions. Every duty has a *wage cost* calculated based on the labour agreement rules. Different types of duty have different governing rules and different methods for calculating the wage costs. The wage cost of an invalid duty could be computed in the same way as that of a valid duty, but the results might not be very meaningful or relevant to the optimisation algorithm. A *penalty* function is defined to reflect how poorly a duty satisfies the governing rules. The penalty of a valid duty is always defined as zero. Drivers and operators may have different preferences to different types of duty. The preferences can be regarded as soft rules and be reflected by weighted cost function (see 4.2.3.2).

4.2.3.1 Duty types

A general discussion of duty types has been provided in Section 1.1.4. Figure 4.3 presents a formal classification defined in this model. In Figure 4.3, A, B and C denote respectively three types of straight duty with at least one mealbreak. They could be *early duty*, *middle duty* and *late duty*, or *ordinary duty*, *shorter duty* and *longer duty* respectively. The name used for other types are self-explanatory. This classification is adequate for normal problem situations, and the types of duty are normally mutually exclusive. In a particular problem, not all types of duty may be required, but at least one, say A in this model, is necessary.

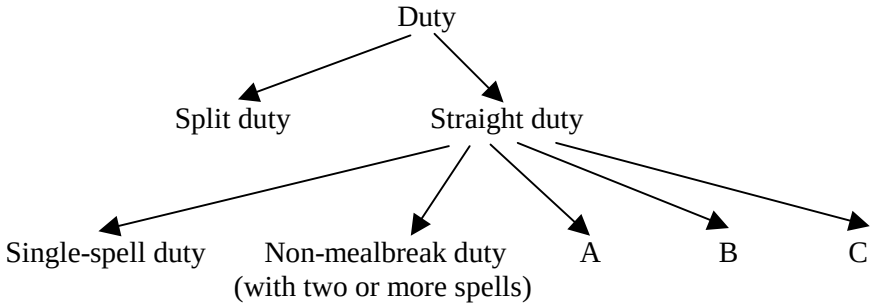


Figure 4.3: Classification of driver duties

Every type of duty required has an associated set of labour agreement rules governing the validity of duties.

4.2.3.2 Costing a duty

The cost and the penalty functions are duty type specific. A *wage cost function* $h(D_i)$, expressed in time units to be paid, can be computed according to the specific method expressed in the labour agreement rules. The following concepts, which are related to the computation of the wage cost function, are defined:

1. *Sign on time* and *sign off time* ---- the shortest time for signing on and off at the same depot, this depot is called the *best-depot* for the duty;
2. *Start on time* and *finish off time* ---- the earliest driving time after signing on and the latest driving time before signing off;
3. *Spreadover* ---- the time duration from signing on to signing off at the depot;
4. *Spell* ---- the driving time of a driver on one vehicle without a break;
5. *Gap* ---- the time between two successive spells;
6. *Join-up* ---- is a gap in which the driver leaves one vehicle and takes over the next vehicle;

7. *Mealbreak* ---- is a gap in which the driver travels to/from the canteen to have a meal;
8. *Sign off/on before and after a long break* ---- the time for the driver to sign off before a long break and to sign on after the break;
9. *Stretch* ---- the time between sign-on to the first mealbreak, or between the end of the last mealbreak to sign-off, or between two consecutive mealbreaks if there are two or more of them; some operators stipulate that the join-up time is excluded from stretches.
10. *Working time* ---- the time counted towards the driver's payment, i.e. the wage cost $h(D_i)$. It is operator and duty type specific and determined by costing rules stipulated in the labour agreement. For instance, the working time could be equal to the spreadover, the spreadover minus a constant (say 30 minutes), or the spreadover minus some unproductive work, e.g. *mealbreak, sign on/off and travel to/from canteen before and after a long break.*

If a duty D_i is valid, the wage cost $h(D_i)$ is defined as being equal to the working time of the duty. In driver scheduling, several types of duty may be allowed, but some types may be more preferable to others. For example, straight duties are often preferred to split duties and non-mealbreak duties. Every type of duty may be assigned an associated subjective weight on its wage cost in order to distinguish the preference. The *cost function* $f(D_i)$ can be defined to reflect the wage cost of the duty D_i and the preference by the following formula.

$$f(D_i) = \begin{cases} h(D_i) * w_1, & \text{if } D_i \text{ is a valid split duty;} \\ h(D_i) * w_2, & \text{if } D_i \text{ is a valid single-spell duty;} \\ h(D_i) * w_3, & \text{if } D_i \text{ is a valid non-mealbreak duty with two or more spells;} \\ h(D_i) * w_4, & \text{if } D_i \text{ is a valid straight duty A;} \\ h(D_i) * w_5, & \text{if } D_i \text{ is a valid straight duty B;} \\ h(D_i) * w_6, & \text{if } D_i \text{ is a valid straight duty C;} \end{cases} \tag{4.1}$$

In Formula 4.1, w_1, w_2, w_3, w_4, w_5, and w_6 are the weights assigned to six different types of valid duty to distinguish their relative preferences. In the UK, straight duties with a mealbreak are usually preferable to the other types of duty. An appropriate assignment to w_1, w_2, w_3, w_4, w_5, and w_6 might use the values 2, 5, 5, 1, 1, 1 respectively. If the weights w_1 to w_6 are all assigned the value of one, there is no preference on any type of valid duty and the cost of a valid duty equals its wage cost. These weights are parameterised and are therefore adjustable for different problems. In the experiments to be presented in Chapter 6, the weights: w_1, w_2, w_3, w_4, w_5, and w_6 are set as 2, 5, 5, 1, 1, 1 respectively. Empirical evidence shows that this set of weights produces better schedules than by setting all the weights to one. The reason may be that the straight duties with a mealbreak are usually more efficient, besides being more preferable, than the split and non-mealbreak duties.

If a duty D_i is invalid, the value of $h(D_i)$ could be computed in the same way but the results might not be very meaningful. For instance, consider a two-spell duty: the sign on and off time allowances are both 5 minutes; the first spell is from 0605 to 1000; the second is from 0610 to 0900. If we calculate the spreadover by subtracting the sign on time from the sign off time, the spreadover of the duty is 3 hours and 5 minutes; we can also calculate the spreadover as the sum of the sign on and sign off time allowances plus the length of the two spells, then the spreadover is 6 hours and 55 minutes. Suppose that the wage cost is defined as the spreadover, the wage cost of this duty is 3:05 or 6:55, but none of the values can reflect the real wage cost of the duty.

A *penalty function* $g(D_i)$ is defined as a value reflecting the infeasibility of the duty D_i, i.e., how poorly D_i conforms to the labour agreement rules. In particular, if D_i is a valid duty then $g(D_i)= 0$. Every labour agreement rule has a corresponding kind of penalty function. If the rule is obeyed, the corresponding penalty is defined as zero, otherwise it is defined as the difference

between the actual value and the corresponding specified limit. For example, if the spreadover of a duty is 8:30 and the maximum length of spreadover for this type of duty is 8:00, the spreadover penalty of this duty is 30 minutes. The formulae given in Section 3.4 are applicable in this model while more formulae need to be defined similarly according to the labour agreement rules. The total penalty $g(D_i)$ is defined as the sum of all types of penalty of the duty D_i.

4.2.3.3 Costing a link

Spell-links constitute the basis for the optimisation algorithm, which requires the definitions of penalty and cost functions of a given link. As already discussed in Section 3.3.3, it is generally not easy to assess precisely how much individual spells and links are contributing to the cost and penalty of a duty. For example, a duty might attract a penalty if there is not a suitable meal break between the third and fifth hour and all duties under eight hours might cost the same. Hence, all the links in the same duty will be assigned the cost and penalty of the duty, i.e., $g(d_{ij}) = g(D_i)$ and $f(d_{ij}) = f(D_i)$ for any link d_{ij} in a duty D_i.

4.2.3.4 Costing a schedule

Given a solution S containing n duties, we define $g(S)$ to be the total penalty of S, and $f(S)$ to be the total cost of S, where

$$g(S) = \sum_{i=1}^{n} g(D_i) \tag{4.2}$$

$$f(S) = \sum_{i=1}^{n} f(D_i) \tag{4.3}$$

4.2.4 Objectives

The goal of driver scheduling is to compile a schedule such that:

- Each piece of work is assigned to a duty;
- Each duty obeys a set of rules, which govern the legality of duties;
- The number of duties required is minimised;
- The total cost of duties is minimised.

According to the definition of a schedule (see Section 4.2.2), the first condition is always satisfied throughout the scheduling process. The last three conditions are the three objectives in our approach. The first objective is to make each duty valid, which is modelled as minimising $g(S)$ until $g(S)= 0$. The second objective is to minimise n, the number of duties in a schedule, which is variable during the scheduling process. The third objective is to minimise total cost, which is modelled as minimising $f(S)$. The first objective is overriding. The second and the third objectives may sometimes be in conflict. A trade-off is therefore needed dependent on the requirement of the operator. In UK, it is common that the second objective has priority over the third one. These two objectives can be combined as minimising:

$$f(S) = \sum_{i=1}^{n} f(D_i) + n * W \tag{4.4}$$

where $W >> f(D_i)$ is a constant, and n is the number of duties in a schedule, which is variable during the scheduling process.

A method exists (Sherali, 1982) which defines *W* as a Sherali weight that guarantees the reduction of duties as a priority. TRACS II uses a Sherali weighted objective function (Fores et al., 1999). The constant chosen in this model is 5000, which adequately prioritises the reduction of duties. The priority will be further guaranteed by rewarding the operations which can reduce the number of duties (see Section 5.6.2).

4.3 An alternative model of a driver schedule

In the previous model, called a *spell-model*, a driver schedule is modelled based on a set of spell-links and their associated AROs. The AROs are determined by how the blocks are cut into spells. Once a set of spells is formed, the AROs are fixed. The current schedule could be improved by breaking up some links and re-forming links in alternative ways. Even though an optimal set of spell-links could be obtained for a given set of spells, the schedule might not be optimal. This is because the initial set of spells and corresponding AROs are usually determined quite roughly. Determining a set of 'good' spells may be as hard as forming a 'good' schedule.

In the spell-model, spells are basic units of driver activities, which can be broken down into smaller units called *pieces of work* (*pieces* for short), which are fixed covering the work between two consecutive WROs in a vehicle. A spell consists of a number of contiguous pieces. Similar to the spell-model, a duty can be represented as a linearly ordered set of *piece-links*. A *piece-link*, which sometimes is called a *link* if it is obviously unambiguous, connects either the sign-on activity to the first piece, two successive pieces, or the last piece to the sign-off activity. Each piece is defined by two ROs in its two WROs respectively (the first RO is called an *arrival-RO* while the second RO is called a *departure-RO*). The formal definition of a schedule *S* to a problem with a block set *B* is presented below, where the notations are the same as those used in the spell-model, but the definitions are slightly different.

$p_{ij} = (t_p, w_p)$	*an arrival-RO,* corresponding to a unique WRO $b_{lk} = (u_{lk}, w_{lk}, v_{lk}) \in B_l \in B$ such that $t_p \in u_{lk}$ and $w_p = w_{lk}$.
$q_{ij} = (t_q, w_q)$	*a departure-RO,* corresponding to a unique WRO $b_{lk} = (u_{lk}, w_{lk}, v_{lk}) \in B_l \in B$ such that $t_q \in u_{lk}$ and $w_q = w_{lk}$.
$d_{ij} = (p_{ij}, q_{ij})$	*a piece-link*, defined by a directed pair of an arrival-RO and a departure-RO.
l_i	*the number of piece-links* in D_i.
$(D_i, \leq) = \{d_{i1} < d_{i2} < \ldots < d_{il_i}\}$	*a duty*, defined by a linearly ordered set of piece-links in the order they occur in the duty. There is an associated function for returning the next link of a given link: $next(d_{ij}) = d_{i,j+1}, \; j=1,2,\ldots,l_i-1$.
n	*the number of duties in the solution.*
$P = \{p_{ij} \mid i = 1,2,\ldots,n; \; j = 1,2,\ldots, l_i\}$	*an arrival-RO set.*
$Q = \{q_{ij} \mid i = 1,2,\ldots,n; \; j = 1,2,\ldots, l_i\}$	*a departure-RO set*.
$S = (P, Q, D)$	*a solution*, defined by a ternary vector, where $D = \bigcup_{i=1}^{n} D_i$.
$\Omega = \{S_i \mid i = 1,2,\ldots,N\}$	*a solution space*, where N is generally an enormous unknown constant in practical problems.

In the above definition, a *schedule* is modelled as a set of *piece-links* with their associated *ROs*. The other factors in driver scheduling such as *vehicle work*, *labour agreement rules* and *objectives* are identically modelled as in the spell-model. Similar to spell-links, all the piece-links in the same duty are assigned the cost and penalty of the duty. This model is called a *piece-model*, in which pieces are fundamental timetabled activities and are fixed. However, the actual driver relief times may be variable within a WRO if it is used.

4.4 Spell-model versus piece-model

We have built a spell-model and a piece-model for the driver scheduling problem with WROs. These two models are analogous; the difference is in the definition of a schedule. Both models define a schedule as a set of links, but one refers to spell-links while the other to piece-links.

Spell-model is built by a natural presentation of the problem, in which a spell is the basic unit of driving work in a duty. During the scheduling process, the compositions of spells are variable. Once an initial schedule is constructed, the set of spells is determined. Breaking up and reforming links may improve the schedule, but is not adequate for finding a good schedule. Even though the optimal combination of the spells could be found, an optimal solution cannot be ensured, because the initial set of spells and the corresponding AROs are usually determined roughly. Re-selecting AROs (i.e., reforming spells) from the entire set of WROs is essential to refining the current schedule. However, unlimited re-selection of AROs would be impractical because there would be too many combinations to consider.

In the piece-model, fixed pieces are the basic units. Finding the optimal combination of pieces by breaking up and reforming the links would lead to an optimal solution. Hence, exchanging links would be the most important operation for improvement. However, exchanging links would be more time-consuming than that in the spell-model because there are many more piece-links than spell-links in a schedule. Moreover, exchanging piece-links would cause resulting duties to have too many fragmented pieces of work.

By combined utilisation of the two models, an enhanced approach would be developed, which is anticipated to allow all ROs a chance of being used while the driver work would not become too fragmented and the computation is efficient.

4.5 Overview of the HACS approach

Based on the models, a heuristic approach for driver scheduling called HACS (Heuristics for Automatic Crew Scheduling) is developed. The fundamental idea of the HACS approach is that of *neighbourhood search* (NS), which first generates an initial schedule and then refines the schedule iteratively. The general scheme of the HACS approach is as follows:

Step 1 *Construct an initial schedule*

Step 2 *Minimise total penalty using a tabu search technique*

Step 3 *Minimise total cost using a tabu search technique without increasing penalty*

If the total cost is reduced and the currently best schedule is still infeasible, go back to step 2; otherwise go to step 4.

Step 4 *Reduce penalty by using extra duties*

If the currently best schedule is feasible, then terminate; otherwise, add an extra duty to reduce total penalty and then go back to step 2.

This scheme attempts to improve the initial solution as much as possible by means of steps 2 and 3. Adding duties is only a last resort. Note that in step 3 (i.e., during minimising cost) the total penalty is not allowed to be increased. Removing penalty is the overriding objective of HACS, which becomes more difficult after Step 2. A feasible schedule is not allowed to be turned into an infeasible schedule during the minimising cost process because the infeasibility cannot be guaranteed to be removed later without using extra duties.

4.5.1 Construction of an initial schedule

An initial solution is the starting point of a heuristic approach and its construction is problem-oriented. Chapter three has presented a simple and quick method to construct an initial schedule for the driver scheduling problem. The initial schedule covers all the vehicle work using a deliberately low target number of duties. The process consists of the following main steps driven by labour agreement rules: first a tight target number of duties is estimated automatically and can be increased or reduced manually; then according to this target number, the block set *B* is partitioned into a set of spells; lastly, the spells are coupled while the last spell, if any remains, forms a one-spell duty. The HACS approach adopts this method to generate an initial solution, which consists of duties with up to two spells. Three and more spell duties may be yielded automatically during the scheduling process.

The initial schedule generated is usually crude and very likely to be infeasible. However, this method is quick, easy to implement, and it is not sensitive to problem variations.

4.5.2 Refinement of the schedule

The initial schedule constructed is very likely to contain infeasible duties. The refining process will improve it iteratively. HACS has three refining process components: *minimising penalty*, *minimising cost*, and *adding duties*. The former two components are driven by the tabu search technique tailored for driver scheduling, which is presented in the next chapter. The major operations in these two minimising processes are *swapping-link* and *recutting block*. Swapping-link improves the schedule by exchanging links while the AROs remain unchanged. Recutting-block improves the schedule by reselecting AROs. Any RO, including those in WROs, has a chance of being selected as an ARO. Hence, the HACS approach could handle WROs. After the

minimising processes using a tabu search, the current best schedule may still be infeasible because the tabu search cannot guarantee an optimal solution, and the target number may be lower than that achievable. Therefore, the *adding-duties* process is devised, analogous to that in the 2-opt approach presented in Section 3.9, to reduce infeasibility until a feasible schedule is obtained:

- One duty is added at a time when the tabu search cannot lead to any improvement.
- Some work from the infeasible duties is taken out to form the extra duty, such that the overall total infeasibility is reduced.

4.6 Conclusions

This chapter has presented two models: *spell-model* and *piece-model* for the driver schedule problem with WROs. The legality of duties according to the labour agreement rules has been quantified by a penalty function while the wage cost and weighted cost functions of duties are defined. These functions can be easily adjusted to different situations. A schedule is modelled as a set of links, which will constitute the basis for the refining operations. There is no limit on the number of spells in a duty as necessitated in the TRACS II system, and WROs could be handled.

This chapter has also presented the general scheme of HACS, a constructive approach based on the models. HACS has four key processes: a) constructing an initial schedule, b) minimising penalty, c) minimising cost, and d) adding duties. The first and the last processes adopt the ideas in the 2-opt heuristic approach. HACS focuses on the processes of minimising penalty and minimising cost, which constitute the core of this research. During the minimising processes, tabu search is employed. The next chapter focuses on the tabu search technique tailored for the driver scheduling problem with WROs.

Chapter Five

Tabu Search Technique Tailored for the Driver Scheduling Problem with WROs

---

5.1 Introduction

Many variations of neighbourhood search algorithms have been proposed based on studies of natural processes, which perform an analogy of optimisation (Aarts and Lenstra, 1997). *Simulated Annealing* (Kirkpatrick et al., 1983; Aarts et al., 1997), *Tabu Search* (Glover, 1989; Glover and Laguna, 1997), *and Genetic Algorithms* (Holland, 1975; Muhlenbein, 1997), are such popular methods and have been labelled as *meta-heuristics* (Reeves, 1993) or *artificial intelligent approaches*. In recent years, these meta-heuristics have attracted strong research interests (see for instance, Pham and Karaboga, 2000; Tian et al., 2000; Mori and Matsuzaki, 2001; Nara et al., 2001). It is also of appeal that they have been widely and successfully used for seeking practical near-optimal solutions to NP-hard problems such as Travelling Salesman

Problem (Reinelt, 1994), Job Shop Scheduling (Brandimarte, 1993; Madureira, 1999), Vehicle Routing (Gendreau et al., 1997; Ozdemir and Mohan, 2000), etc.

An online computing dictionary (http://www.instantweb.com/d/dictionary/index.html) gives meta-heuristics the following definition:

> *A top-level general strategy which guides other heuristics to search for feasible solutions in domains where the task is hard.*

Guiding the 2-opt heuristics, Tabu Search (Glover, 1986, 1989 and 1990) is investigated for the driver scheduling problem with WROs. Glover and Laguna (1993) indicate that a fundamental element underlying Tabu Search is the use of flexible memory, which creates and exploits structures to take advantage of history. Strategic use of memory can make dramatic differences in the ability to solve problem (Glover and Laguna, 1997), but Tabu Search is a general search scheme that must be tailored to the details of the problem at hand. Unfortunately, there is little theoretical knowledge that guides this tailoring process, and users have to resort to the available practical experience.

This chapter first introduces the Tabu Search method, and then presents the main components of the Tabu Search technique tailored for the driver scheduling problem with WROs, in which objective functions are defined while multi-neighbourhoods and an appropriate memory scheme are devised. Before the Conclusions, the Tabu Search algorithms in HACS are outlined. The driver scheduling problem without WROs is a special case of that with WROs. Much of the work in this chapter has been published by Shen and Kwan (2001).

5.2 Tabu Search

Tabu Search (TS) was first proposed by Glover (1986) although its roots go back to the 1970s. Hansen (1986) then sketched the basic idea. Additional efforts of formalization were reported by Glover (1989, 1990) and de Werra & Hertz (1989). A comprehensive account of the basic concepts, and of recent developments was given by Glover and Laguna (1993). Later Glover and Laguna (1997) provided "hands-on" knowledge of TS and insight alike by presenting the major ideas of TS with examples that showed their relevance to multiple applications. New applications of TS are emerging frequently (i.e., Tian et al., 2000; Mori and Matsuzaki, 2001). The fundamental idea of TS is that of neighbourhood search. To avoid being trapped in local optima, TS allows the acceptance of non-improved solutions. In this way, the search is directed away from local optima, such that other parts of the search space can be explored. To prevent going back to recently visited solutions, a memory scheme is used to record the moves made in the recent past of the search. This recorded search history is usually denoted by H and presented by a *tabu list* of moves, which are forbidden for a certain number of iterations. However, a move labelled tabu is still acceptable if a certain *aspiration criterion* is satisfied, for instance, the move leads to a good enough solution. With the search history record H, the neighbourhood $N(x^{now})$ is modified to $N(H, x^{now})$. This history is dynamically updated during the search process. As a more elaborate neighbourhood search method, TS can be initially described as a *steepest descent method* with a *dynamic neighbourhood* associated with an exploration history. However, since finding a best solution in $N(H, x^{now})$ may often be too time-consuming, it is usually crucial to determine a *candidate set* as a subset of $N(H, x^{now})$. Table 5.1 shows a basic TS algorithm.

Table 5.1: Tabu Search

Step 1	(Initialisation) Get an initial solution x^{now}; Record the current best solution by setting $x^{best} = x^{now}$; Set history record $H = \varnothing$.
Step 2	(Move choice and termination) Determine $Candidate\_N(x^{now})$ as a subset of N (H, x^{now}); Choose a best solution $x^{next} \in Candidate\_N(x^{now})$ such that $c(x^{next}) \leq c(x)$ for any $x \in Candidate\_N(x^{now})$; If no solution qualifies to be x^{next} or other termination criteria apply, the method stops.
Step 3	(Update) Set $x^{now} = x^{next}$; Set $x^{best} = x^{now}$ if x^{now} is better than the current best solution; Update H; Then go to Step 2.

TS has been applied to a large variety of problems with considerable success (see for instance Hertz and de Werra, 1991; Mori and Matsuzaki, 2001), although no clean proof of convergence is known and there has been no formal explanation of this good behaviour up to now (Hertz, Taillard and de Werra, 1997). There are investigations of theoretical aspects of TS (e.g. Faigle and Kern, 1992), but this research mainly concentrates on its tailoring for the driver scheduling problem.

5.3 Objective functions

The TS technique is employed in two phases: *minimising penalty* and *minimising cost* (see the general scheme in Section 4.5), in which the objective functions are defined (in Section 4.2.4) as $min(g(S))$ and $min(f(S))$ respectively, and in the phase of minimising cost, the total penalty is not allowed to be increased.

5.4 Multi-neighbourhoods based on the spell model

Tabu Search may be conveniently characterised as a form of neighbourhood search (see Glover and Laguna, 1993), where each solution $S \in \mathcal{S}$ has an associated set of neighbours $N(S) \subset \mathcal{S}$ called the *neighbourhood* of *S*. Any solution $S' \in N(S)$ can be reached directly from *S* to *S'* by an operation called a *move*. 2-opt exchange is frequently used to define neighbourhoods in permutation problems, e.g. TSP and filtering sequence problems (Glover and Laguna, 1993). Applying 2-opt exchange and its variants to the HACS approach, neighbourhoods for the driver scheduling problem are defined by the following move operations: *swapping-two-links*, *swapping-two-spells*, *insert-one-spell*, and *recutting-block*. More than one move operations (i.e. multi-neighbourhoods) are employed in HACS, the reason is provided while defining the operations below, where D_1 and D_2 denote two duties, and $D_1 = \{d_{11}, d_{12}, \ldots, d_{1i}, \ldots, d_{1s}\}$, where *s* is the number of links in D_1; $D_2 = \{d_{21}, d_{22}, \ldots, d_{2j}, \ldots, d_{2t}\}$, where *t* is the number of links in D_2.

5.4.1 Swapping two links

This is a typical 2-opt swapping operation. Given two links $d_{1i} = (p_{1i}, q_{1i}) \in D_1$, $d_{2j} = (p_{2j}, q_{2j}) \in D_2$, the operation *swapping-two-links*(d_{1i}, d_{2j}) is defined as follows.

If $D_1 = D_2$, this operation will do nothing, otherwise, it operates by the following steps:

Step 1: *Exchange-link*$_1$ $(\{d_{1i}, d_{2j}\}) = \{ d'_{1i}, d'_{2j}\}$, where

$d'_{1i} = (p_{1i}, q_{2j})$, $d'_{2j} = (p_{2j}, q_{1i})$.

Step 2: *Exchange-duty*$_1$ $(\{D_1, D_2\}) = \{ D'_1, D'_2\}$, where

$D'_1 = \{d_{11}, d_{12}, \ldots, d_{1,i-1}, \mathbf{d'_{1i}}, d_{2,j+1}, \ldots, d_{2t}\}$,

$D'_2 = \{d_{21}, d_{22}, \ldots, d_{2,j-1}, \mathbf{d'_{2j}}, d_{1,i+1}, \ldots, d_{1s}\}$.

Step 3: $d_{1i} = d'_{1i;}\ d_{2j} = d'_{2j;}\ D_1 := D'_{1;}\ D_2 := D'_{2.}$

This operation replaces the given links d_{1i}, d_{2j} with two new links d'_{1i}, d'_{2j}. Figure 5.1 provides an illustration, where a thick line denotes a spell, a solid arrow denotes an original link and a dotted arrow denotes a new link generated by the operation.

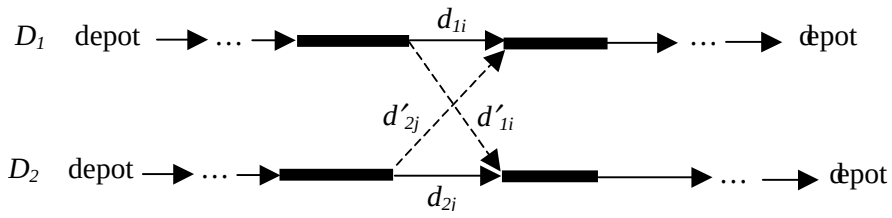


Figure 5.1: Illustration of swapping two links

In the 2-opt heuristic approach, we have considered all possible scenarios of 2-opt exchange (see Section 3.7). That is not practical for HACS, in which the number of links in a duty is unlimited. In addition, it is not necessary because any spell could be changed to any position in a duty after the operation, i.e., any spell could be changed into a start spell, an end spell, or a spell in the middle of a duty. Figure 5.2 shows an example, in which the spell denoted by the thickest line will become an end spell while the end spell in D_2 will become a spell in the middle of a duty after the swapping-two-links operation.

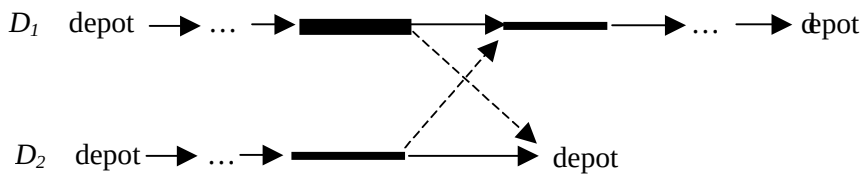


Figure 5.2: Example of turning a spell into an end spell

Despite this, it is hard for the swapping-two-links operation to generate all possible links because each link is tied by the whole chain and there are many restrictions on the duty generation. Two other swapping operations have been designed to provide more chances of refining the solution. Later experiments show that the use of these three swapping operations together produces better solutions than the use of a single swapping operation.

5.4.2 Swapping two spells

Given two links $d_{1i} = (p_{1i},q_{1i}) \in D_1$, $d_{2j} = (p_{2j},q_{2j}) \in D_2$, the operation *swapping-two-spells*(d_{1i},d_{2j}) is defined as follows.

If $D_1 \neq D_2$ and none of the links is a sign-off activity, i.e. there exist $d_{1,i+1} = (p_{1,i+1}, q_{1,i+1}) \in D_1$, $d_{2,j+1} = (p_{2,j+1}, q_{2,j+1}) \in D_2$ such that $next(d_{1i}) = d_{1,i+1}$ and $next(d_{2j}) = d_{2,j+1}$, the operation will be done by the following steps, otherwise nothing will be done.

Step 1: $Exchange\text{-}link_2(\{d_{1i}, d_{2j}\}) = \{d'_{1i}, d'_{2j}, d'_{1,i+1}, d'_{2,j+1}\}$, where

$d'_{1i} = (p_{1i}, q_{2j})$, $d'_{2j} = (p_{2j}, q_{1i})$,

$d'_{1,i+1} = (p_{1,i+1}, q_{2,j+1})$, $d'_{2,j+1} = (p_{2,j+1}, q_{1,i+1})$.

Step 2: $Exchange\text{-}duty_2(\{D_1, D_2\}) = \{D'_1, D'_2\}$, where

$D'_1 = \{d_{11}, d_{12}, \ldots, d_{1,i-1}, \boldsymbol{d'_{1i}, d'_{2,j+1}}, d_{1,i+2}, \ldots, d_{1s}\}$,

$D'_2 = \{d_{21}, d_{22}, \ldots, d_{2,j-1}, \boldsymbol{d'_{2j}, d'_{1,i+1}}, d_{2,j+2}, \ldots, d_{2t}\}$.

Step 3: $d_{1i} = d'_{1i}$; $d_{2j} = d'_{2j}$; $d_{1,i+1} = d'_{1,i+1}$; $d_{2,j+1} = d'_{2,j+1}$; $D_1 := D'_1$; $D_2 := D'_2$.

This operation replaces the given two links $d_{1i,}\ d_{2j}$ and their successors $d_{1,i+1,}\ d_{2,j+1}$, if they exist, with four new links $d'_{1i,}\ d'_{2j,}\ d'_{1,i+1,}$ and $d'_{2,j+1}$ as illustrated in Figure 5.3, where the notations have the same meanings as those used in Figure 5.1.

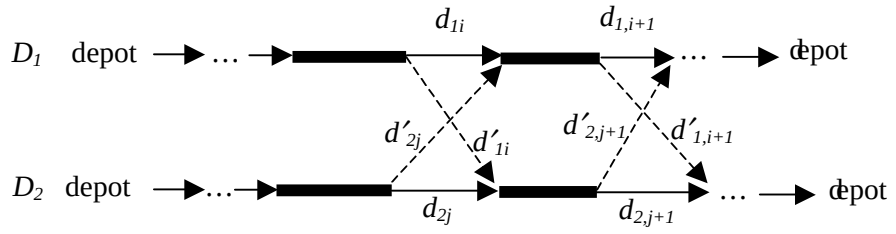


Figure 5.3: Illustration of swapping two spells

5.4.3 Inserting one spell

Given two links $d_{1i} = (p_{1i},q_{1i}) \in D_1$, $d_{2j} = (p_{2j},q_{2j}) \in D_2$, the operation *inserting-one-spell*(d_{1i},d_{2j}) is defined as follows.

If $D_1 \neq D_2$ and the link d_{1i} is not a sign-off activity, i.e. there exists $d_{1,i+1} = (p_{1,i+1}, q_{1,i+1}) \in D_1$ such that $next(d_{1i}) = d_{1,i+1}$, the operation will be done by the following steps, otherwise, nothing will be done.

Step 1: $Exchange\text{-}link_3\ (\{d_{1i}, d_{2j}\}) = \{d'_{1i,}\ d'_{2j,}\ d'_{1,i+1}\}$, where

$d'_{1i} = (p_{2j}, q_{1i})$, $d'_{2j} = (p_{1i}, q_{1,i+1})$, $d'_{1,i+1} = (p_{1,i+1}, q_{2j})$.

Step 2: $Exchange\text{-}duty_3\ (\{D_{1,}\ D_2\}) = \{D'_{1,}\ D'_2\}$, where

$D'_1 = \{d_{11,}\ d_{12,}\ldots,\ d_{1,i-1,}\ \boldsymbol{d'_{2j,}}\ d_{1,i+2,}\ldots,\ d_{1s}\}$,

$D'_2 = \{d_{21,}\ d_{22,}\ldots,\ d_{2,j-1,}\ \boldsymbol{d'_{1i,}}\ \boldsymbol{d'_{1,i+1,}}\ d_{2,j+2,}\ldots,\ d_{2t}\}$.

Step 3: $d_{1i} = d'_{1i;}\ d_{2j} = d'_{2j;}\ d_{1,i+1} = d'_{1,i+1;}\ D_1 := D'_{1;}\ D_2 := D'_2$.

This operation replaces the given two links d_{1i}, d_{2j}, and the link $d_{1,i+1}$ (the successor of d_{1i}), if it exists, with three new links $d'_{1i,}$ $d'_{2j,}$ and $d'_{1,i+1}$ as illustrated in Figure 5.4, where the notations have the same meanings as those used in Figure 5.1.

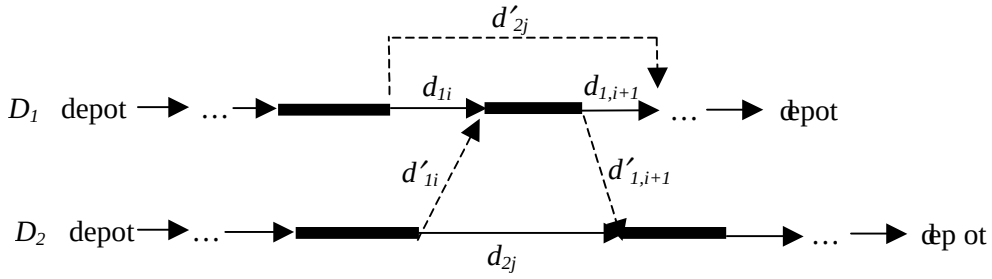


Figure 5.4: Illustration of inserting one spell

Note that the operation *inserting-one-spell*(d_{1i},d_{2j}) is different from the operation *inserting-one-spell*($d_{2j,}d_{1i}$). The latter operation inserts the spell following the link d_{2j} into the link d_{1i}.

5.4.4 Recutting block

In the spell-based model, spells are the basic units of driver duties. The initial set of spells and corresponding AROs are determined quite roughly. Recutting blocks is therefore necessary and plays an important role in refining the current schedule by re-selecting AROs (i.e. re-compose spells).

5.4.4.1 Definition of the potential alternatives of a given ARO

In the process of recutting blocks (i.e. re-selecting AROs), the most important task is to define which ROs are to replace the current AROs. In HACS, only the ROs adjacent to the current AROs are considered as potential AROs, i.e. the predecessor and the successor of an ARO are

defined as candidates. If this restriction were relaxed, the process would be impractical because there would be too many combinations to consider. However, according to this stipulation, each RO still has potential opportunities to be an ARO. The reason is as follows: Once the predecessor or the successor of a current ARO is selected as an ARO, the predecessor or the successor of the new selected ARO has chance to be an ARO, and so on. Any RO could be an ARO.

Further, we stipulate that an arrival-ARO may be replaced by its nearest previous RO (predecessor) while a departure-ARO may be replaced by its nearest next RO (successor). This stipulation is consistent with the previous one. The reason is as follows:

- If a RO is at the beginning of a block, it must be a departure-ARO and only has a successor.
- Similarly, if a RO is at the end of a block, it must be an arrival-ARO and only has a predecessor.
- If an ARO is in the middle of a block, it must be included in two different spells. In one spell it is an arrival-ARO having a predecessor while in another spell it is a departure-ARO having a successor.

5.4.4.2 Definition of the predecessor and successor of an ARO without considering WROs

If WROs are not considered, a block $(B_i, \leq) = \{b_{i1} < b_{i2} < \ldots \lesssim b_{im_i}\}$ can be simplified as $(B_{i,} \leq) = \{r_{i1} < r_{i2} < \ldots \lesssim r_{im_i}\}$, where r_{ij} is a RO, a special case of a WRO. Any ARO must be a RO r_{ij} in a certain block $B_{i.}$ Therefore we can define the functions *predecessor* and *successor* as:

$$predecessor(r_{ij}) = r_{i,j-1,}\ 1 < j \leq m_{i,} \quad \text{where } r_{ij} \text{ is a arrival-ARO.} \tag{5.1}$$

$$successor(r_{ij}) = r_{i,j+1,}\ 1 \leq j < m_{i,} \quad \text{where } r_{ij} \text{ is a departure-ARO.} \tag{5.2}$$

5.4.4.3 Conversion of a time window into discrete times

Before defining the predecessor and the successor of an ARO considering WROs, we convert time windows into discrete times.

A WRO consists of a range of times. Since the HACS approach handles ROs as discrete times, a possible way to handle WROs is to expand explicitly each time window into individual minutes. Considering the difference between A-WROs and U-WROs, two different conversions are devised as follows:

In an *A-WRO* a driver must be on duty but can be relieved by another. Therefore, an A-WRO can be modelled as a sequence of relief opportunities with one-minute intervals. For example, supposing $b_{ij} = (u_{ij,}\ w_{ij,}\ v_{ij})$ is an A-WRO and $u_{ij} = [t_{1,}\ t_2]$, then b_{ij} can be treated as a sequence of ROs: $r_{1,}\ r_{2,..,}\ r_{m,}$ where $m = t_2 - t_1 + 1$, and $r_k = (t_1 + k - 1,\ w_{ij})$, $k \in \{1,2,\ldots,m\}$.

In contrast with an A-WRO, an *U-WRO* requires no driver. The usual practice, e.g. in TRACS II, is to either select only the arrival time and the departure time as relief times or divide a block containing several large gaps of U-WROs into several shorter blocks. This conversion does not compromise the property of U-WROs because the times in between U-WROs do not contribute to improving the schedule, and in contrast, they increase unproductive driver work. Consequently, HACS also selects only the arrival time and the departure time as relief opportunities. However, It is not necessary for HACS to split blocks containing large gaps of U-WROs. HACS stipulates that the arrival time is applied when the U-WRO is at the end of the spell while the departure time is applied when the U-WRO is at the beginning of the spell.

5.4.4.4 Enhanced definitions of the predecessor and successor of an ARO

Based on the foregoing conversion of WROs into discrete ROs, the functions *predecessor*(p_{ij}) and *successor*(q_{ij}) involving WROs are defined below, by reference to Figure 5.5 and the notations in Sections 4.2.1 and 4.2.2.

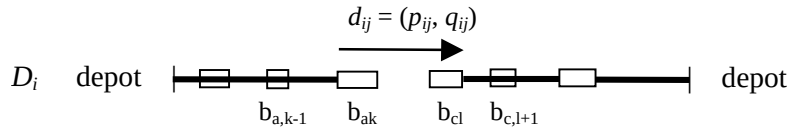


Figure 5.5: Duty with a selected link and its WROs

In Figure 5.5, $d_{ij} = (p_{ij}, q_{ij})$ is a link in a duty $D_{i.}$ Suppose $p_{ij}= (t_p,w_p)$, $q_{ij}=(t_q,w_q)$ corresponding to $b_{ak} = (u_{ak,} w_{ak,} v_{ak}) \in B_a$ and $b_{cl} = (u_{cl,} w_{cl,} v_{cl}) \in B_c$ respectively, where $t_p \in u_{ak} =[t_{ak1},t_{ak2}]$, $t_q \in u_{cl} =[t_{cl1},t_{cl2}]$, and $w_p= w_{ak}$, $w_q= w_{cl}$. If b_{ak} is an U-WRO, then $v_{ak} = 0$, otherwise, $v_{ak} = 1$. If b_{cl} is an U-WRO, then $v_{cl} = 0$, otherwise, $v_{cl} = 1$. We define the predecessor of p_{ij} and the successor of q_{ij} as follows.

$$\text{predecessor}(p_{ij}) = \begin{cases} (t_p\text{-}1,w_p), \text{ if } v_{ak} = 1,\ t_p > t_{ak1}; \\ (t_{ak1},w_p), \text{ if } v_{ak} = 0,\ t_p > t_{ak1}; \\ (t_{a,k\text{-}1,2},w_p), \text{ if } v_{a,k\text{-}1} = 1,\ t_p = t_{ak1}; \\ (t_{a,k\text{-}1,1},w_p), \text{ if } v_{a,k\text{-}1} = 0,\ t_p = t_{ak1}; \end{cases} \tag{5.3}$$

$$\text{successor}(q_{ij}) = \begin{cases} (t_q + 1,w_p), \text{ if } v_{cl} = 1,\ t_q < t_{cl2}; \\ (t_{cl2},w_p), \text{ if } v_{cl} = 0,\ t_q < t_{cl2}; \\ (t_{c,l+1,1},w_p), \text{ if } v_{c,l+1} = 1,\ t_q = t_{cl2}; \\ (t_{c,l+1,2},w_p), \text{ if } v_{c,l+1} = 0,\ t_q = t_{cl2}; \end{cases} \tag{5.4}$$

These new definitions are also applicable to the problems without time windows, in which $t_p= t_{ak1} = t_{ak2}$, $t_q= t_{cl1} = t_{cl2}$ and $(t_{a,k-1,1,} w_p) = (t_{a,k-1,2,} w_p)$, $(t_{c,l+1,1,} w_p) = (t_{c,l+1,2,} w_p)$, i.e. the formulae (5.3) and (5.4) embody the formulae (5.1) and (5.2) respectively.

5.4.4.5 Definition of recutting-block operation

Given a link $d_{1i} = (p_{1i},q_{1i}) \in D_1$, where $p_{1i} \in B_1$, $q_{1i} \in B_2$, and $D_1 = \{d_{11},d_{12},\ldots,d_{1i},\ldots,d_{1s}\}$, where s is the number of links in D_1, if there exists a link $d_{2j} = (p_{2j}, q_{2j})$ such that

- $d_{2j} \in D_2 = \{d_{21},d_{22},\ldots,d_{2j},\ldots,d_{2t}\}$, where t is the number of links in D_2,
- $D_1 \neq D_2$,
- $p_{1i} = q_{2j}$ or $p_{2j} = q_{1i}$,

The link d_{2j} is called the *relevant link* of d_{1i}. We then define the operation *recutting-block* by cases as follows:

Case 1: $p_{1i} = q_{2j}$

Step 1: Search for a component r_{1v} in B_1 such that $predecessor(p_{1i}) = r_{1v}$.

Step 2: $Exchange\text{-}link_4(\{d_{1i}, d_{2j}\}) = \{d'_{1i}, d'_{2j}\}$, where $d'_{1i} = (r_{1v}, q_{1i})$, $d'_{2j} = (p_{2j}, r_{1v})$.

Step 3: $Exchange\text{-}duty_4(\{D_1, D_2\}) = \{D'_1, D'_2\}$, where

$$D'_1 = \{d_{11}, d_{12},\ldots, d_{1,i-1}, \boldsymbol{d'_{1i}}, d_{1,i+1},\ldots, d_{1s}\},$$

$$D'_2 = \{d_{21}, d_{22},\ldots, d_{2,j-1}, \boldsymbol{d'_{2j}}, d_{2,j+1},\ldots, d_{2t}\}.$$

Step 4: Remove p_{1i} from P.

Step 5: Remove q_{2j} from Q.

Step 6: Put r_{1v} into P,Q.

Step 7: $d_{1i} = d'_{1i}$; $d_{2j} = d'_{2j}$; $D_1 := D'_1$; $D_2 := D'_2$.

Case 2: $q_{1i} = p_{2j}$

Step 1: Search for a component r_{1v} in B_2 such that $successor(q_{1i}) = r_{1v}$.

Step 2: $Exchange\text{-}link_4(\{d_{1i}, d_{2j}\}) = \{d'_{1i}, d'_{2j}\}$, where $d'_{1i} = (p_{1i}, r_{1v})$, $d'_{2j} = (r_{1v}, q_{2j})$.

Step 3: $Exchange\text{-}duty_4(\{D_1, D_2\}) = \{D'_1, D'_2\}$, where

$D'_1 = \{d_{11}, d_{12}, \ldots, d_{1,i-1}, \boldsymbol{d'_{1i}}, d_{1,i+1}, \ldots, d_{1s}\}$,

$D'_2 = \{d_{21}, d_{22}, \ldots, d_{2,j-1}, \boldsymbol{d'_{2j}}, d_{2,j+1}, \ldots, d_{2t}\}$.

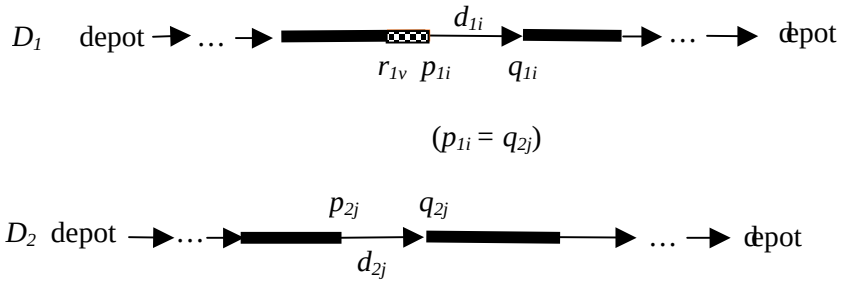
Step 4: Remove q_{1i} from P.

Step 5: Remove p_{2j} from Q.

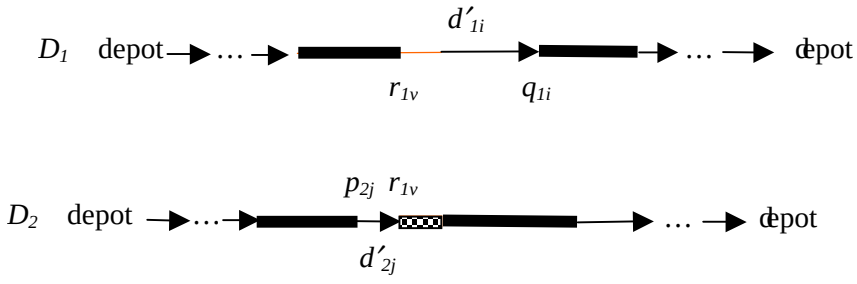
Step 6: Put r_{1v} into P and Q.

Step 7: $d_{1i} = d'_{1i}$; $d_{2j} = d'_{2j}$; $D_1 := D'_1$; $D_2 := D'_2$.

An illustration of *recutting-block* of case 1 is given in Figure 5.6. Figure 5.6(a) shows two original duties while Figure 5.6(b) shows the corresponding new duties, where the piece of work presented as a hatched pattern is removed from D_1 and then inserted into D_2 after the recutting-block operation.



(a) Two original duties



(b) Two new duties

Figure 5.6: Illustration of recutting blocks

5.4.5 Summary

Four move operations (corresponding to multi-neighbourhoods) have been defined in this section. Given the current solution $S = (P,Q,D)$, the first three operations transform $S=(P,Q,D)\in \Ş$ to $S' =(P,Q,D)\in \Ş$, where the sets of AROs P and Q remain unchanged. These three operations can be categorised as *swapping-link* (or *swapping* for short) operations denoted as *swapping*(d_{1i}, d_{2j}) because they involve swapping several links triggered by a given pair of links (d_{1i}, d_{2j}) while the set of AROs remains unchanged. The recutting-block operation is different from the swapping operations. It revises the selection of AROs, i.e. P, Q and D are changed when the move transforms a solution $S=(P,Q,D)\in \Ş$ to its neighbour $S' =(P' ,Q' ,D')\in \Ş$.

Any link d_{1i} and its relevant link d_{2j}, which is uniquely determined by d_{1i}, can trigger a unique recutting-block move, hence we can denote it as *recutting*(d_{1i}, d_{2j}). Referring to a non-specific move, the move is denoted as *move*(d_{1i}, d_{2j}) instead of *swapping*(d_{1i}, d_{2j}) or *recutting*(d_{1i}, d_{2j}).

5.5 Multi-neighbourhoods with combined utilisation of the spell model and the piece model

The previous section has defined three swapping-link operations and a recutting-block operation. In these operations, the spells are treated as basic units. In addition to swapping links, recutting blocks plays an important role in refining the current schedule because the initial set of spells and corresponding AROs were determined quite roughly. However, the recutting function is limited because only the ROs adjacent to the current AROs are considered as potential AROs. If this restriction were relaxed, the process would be impractical because there would be too many combinations to consider.

In the above swapping-link and recutting-block operations, spells are the basic units of driver work. By considering the *piece of work* (piece for short) covered between two successive ROs as a more basic unit, a spell consists of a number of pieces contiguously linked together. The four operations could be applied to the links between the pieces instead of between spells. The new piece-based operations are *swapping-two-links, swapping-two-pieces, inserting-one-piece,* and *recutting-block,* which are analogous to the spell-based operations accordingly. The piece-based new operations can be identically illustrated by Figures 5.1, 5.3, 5.4 and 5.6 respectively, but the thick lines denote pieces instead of spells.

In the piece-based model, pieces of work are fixed indivisible units. During the swapping-link operations, each WRO has a chance of being tried as an active WRO. Hence, the piece-based swapping operations can be used to replace the spell-based recutting-block operation. The limitation that only the ROs adjacent to the current AROs are considered as potential AROs in the spell-based recutting-block operation is thus removed.

When WROs are considered, an ARO may be replaced by another RO in the same WRO. The recutting-block operation is therefore retained to re-select an ARO from within its corresponding WRO. Limiting the recutting-block operation within WROs, the definitions of the predecessor of p_{ij} and the successor of q_{ij} in the formulae 5.3 and 5.4 (see Section 5.4.4.4) should be changed as follows.

$$\text{predecessor}(p_{ij}) = \begin{cases} (t_p\text{-}1, w_p), \text{ if } v_{ak} = 1,\ t_p > t_{ak1}; \\ (t_{ak1}, w_p), \text{ if } v_{ak} = 0,\ t_p > t_{ak1}; \end{cases} \tag{5.5}$$

$$\text{successor}(q_{ij}) = \begin{cases} (t_q + 1, w_p), \text{ if } v_{cl} = 1,\ t_q < t_{cl2}; \\ (t_{cl2}, w_p), \text{ if } v_{cl} = 0,\ t_q < t_{cl2}; \end{cases} \tag{5.6}$$

When no WROs are involved, the piece-based swapping-link operations would be adequate and the recutting-block operation is therefore not applied. Unfortunately, the piece-based swapping

operations would be very time-consuming, and the resulting duties would have too many fragmented pieces of work. A compromise would be a combination of the following operations: spell-based *swapping-two-links*, *swapping-two-spells* and *inserting-one-spell*; piece-based *swapping-two-pieces* and *inserting-one-piece*; and *recutting-block* for WROs.

The HACS approach incorporating the spell-model and the piece model (i.e., applying the five swapping operations and a recutting-block operation for WROs) is anticipated to be superior to the HACS approach based on a single model, because all ROs would have a chance of being used while the driver work would not become too fragmented.

5.6 Memory schemes

The aggressive aspect of TS is manifest in choice rules that seek the best available move that can be determined with an appropriate amount of effort (see Glover and Laguna, 1997). The meaning of 'best' in TS applications is customarily not limited to an objective function evaluation. In HACS, the *best move* is defined as the move with a highest move value amongst the moves either being non-tabu in the neighbourhood or satisfying a certain *aspiration criterion*, which determines when tabu activation rules can be overridden. In HACS, we employ the *improved-best* criterion, i.e., removing a tabu classification from a trial move when the move yields a solution better than the best obtained so far.

For many situations, where the neighbourhood $N(S,H)$ (which depends on the current solution S and the search history H) is large or its elements are expensive to evaluate, finding the best move in $N(S,H)$ may often be too time-consuming. Cutting down the computational effort is vital. Before addressing the methods for cutting down the computational effort, some basic components of TS, e.g., *move attribute*, *move value*, *tabu tenure* and *tabu status* are defined.

5.6.1 Move representation and move attribute

The fundamental operation in TS is called a *move*. The *move operations* used in HACS have been defined in Sections 5.4 and 5.5. According to the definitions of the move operations, a link (a spell-link or a piece-link) is the basic unit of a move operation. Given a solution, a set of links is fixed and the number of links is known. We store all the links in a vector $V = (d_1, d_2, \ldots, d_k)$, where k is the number of links in the solution. Each link corresponds to a unique index in the vector, hence we can use the index to denote a link, and a move *swapping*(d_i, d_j) or *recutting*(d_i, d_j) can be represented as a *move*(i,j), where $i,j \in \{1,2,\ldots,k\}$.

Each move has an associated *attribute* that can encompass any aspect that changes as a result of the move. After a move, indices for the original pair of links are reused for the new pair of links. Therefore, the attribute of a *move*(i,j) can be defined as the pair (i,j), in which the links d_i and d_j change from the original ones to the current ones.

Recorded move attributes are often used to impose tabu restrictions that prevent moves from being selected that would reverse the changes represented by these attributes. In the swapping operations, a *move*(i,j) is tabu if the *move*(i,j) has been executed previously. In the recutting operation, a *move*(i,j) is tabu if the *move*(j,i) has been previously executed.

5.6.2 Move value

Move values are used to measure moves. The value of a *move*(i,j) can be defined as $f(S) - f(S')$ or $g(S) - g(S')$, where S is the current schedule and S' is the resultant schedule of the move, depending on the phase of the algorithm. According to this definition, a preferable move has a

high value. The move values are updated from one iteration to another. Efficient updating of move values can reduce the effort of finding the best or near best moves. In driver scheduling, the move values can be calculated without a full evaluation of the objective function as follows:

Obviously, only the duties altered by the move need to be evaluated. Since we have defined the cost and penalty of a link to be those of the duty it belongs to (see Section 4.2.3.3), the move value $f(S) - f(S')$ or $g(S) - g(S')$ can be equivalently represented as $f(d_i) + f(d_j) - f(d'_i) - f(d'_j)$ while $g(S) - g(S')$ can be equivalently represented as $g(d_i) + g(d_j) - g(d'_i) - g(d'_j)$.

As addressed in Section 4.2.4, the objective of minimising the total number of duties usually has priority over the objective of minimising cost. If a move can lead to the reduction of the number of duties without increasing infeasibility, we will prioritise the move by adding a large constant, say 5000, to the value of the move. Moreover, if a move is prohibited, e.g. both links of a move are in the same duty, the move will be assigned a very low value, say –5000.

5.6.3 Tabu tenure and tabu status

As already mentioned, the basic idea of TS is to allow the acceptance of non-improved solutions to avoid being trapped in local optima. To prevent going back to recently visited solutions, a memory is used to record the moves made in the recent past of the search. This memory is called a *tabu list*, in which the recorded moves are forbidden for a certain number of iterations. The certain number of iterations for which a move must remain tabu is called the *tabu status* of the move. Each move has a corresponding *tabu status*. If a move is not tabu, its tabu status is usually defined as zero; otherwise its tabu status depends on the *tabu tenure* and the current iteration. The *tabu tenure* decides the number of iterations for which the current executed move must remain tabu. Therefore, the tabu status of a current executed move can be

defined as the tabu tenure, and then at each subsequent iteration the tabu status of the move is subtracted by one, continuing until the tabu status is zero. There are other ways (see Section 5.6.4.2 for instance) to define the tabu status of a move but the meaning is similar.

No single rule has been designed to yield an effective tenure for all classes of problems (Glover and Laguna, 1997). However, Glover and Laguna (1993) indicate that values between 7 and 20 appear to work well for a variety of problem classes, while values between $0.5\sqrt{n}$ and $2\sqrt{n}$ appear to work well for other classes, where n is a measure of problem dimension. In HACS, $min(\sqrt{n}, 20)$ is defined as the tabu tenure, where n denotes the number of spell-links, which equals the sum of the number of spells and the number of duties in the current schedule. Experiments have shown that using this variable function is more effective than using a constant between 7 and 20, although the usual range for this function is between 7 and 20 for practical problems. For example, considering practical driver scheduling problems with about 100 duties, which are mostly consisting of two or three spells (i.e. 3 or 4 spell-links per duty), $\sqrt{n}$ is usually less than 20. Considering small problems with 14 duties $\sqrt{n}$ is approximately equal to 7.

5.6.4 Candidate list and tabu list

As already addressed, for many situations, where the neighbourhood $N(S,H)$ is large or its elements are expensive to evaluate, finding the best move in $N(S,H)$ may often be too time-consuming. Cutting down the computational effort is vital. The usual method is to determine a *candidate set* as a subset of $N(S, H)$ to restrict the number of solutions examined on a given iteration. The importance of determining a candidate set has been described a great deal in the literature. Glover and Laguna (1997) suggest some general classes of candidate set (or

candidate list) strategies, such as *aspiration plus*, *elite candidate list*, etc. Employing the elite candidate list strategy, a memory scheme is designed for the driver scheduling problem.

5.6.4.1 Elite candidate list

The Elite Candidate List approach (Glover and Laguna, 1997) first builds a candidate list by examining all moves, selecting the *k* best moves encountered, where *k* is a parameter. Then at each subsequent iteration, the current best move from the candidate list is selected to be executed. This process terminates when such a move falls below a given quality threshold, or until a given number of iterations have elapsed. Then a new elite candidate list is constructed and the process repeats. As discussed by Glover and Laguna (1997), this method is created based on the assumption that a good move, if not performed at the present iteration, will still be a good move for some number of iterations. This assumption might be true in many situations, but our experiments have shown that this assumption is not appropriate for the HACS driver scheduling approach. The explanation is given by the following example.

In the experiments, an elite candidate list is designed for driver scheduling using the above-described method. The elite candidate list first records the best *k* moves, where the parameter *k* is arbitrarily defined to be equal to thirty. Suppose the current best move is *move*(*i*,*j*). The move *move*(*i*,*x*) may be also recorded in the candidate list. Figure 5.7 illustrates an example, in which *link i*, *link j* and *link x* are the invalid links involved in *move*(*i*,*j*) and *move*(*i*,*x*). Both moves *move*(*i*,*j*) and *move*(*i*,*x*) can make a considerable improvement on the schedule, such that they are selected into the elite candidate list. The improvement can be seen by comparing Figure 5.7 and Figure 5.8, where the new links after the moves are shown. Suppose *move*(*i*,*j*) is executed, *link i* changes and the quality of *move*(*i*,*x*) becomes poor.

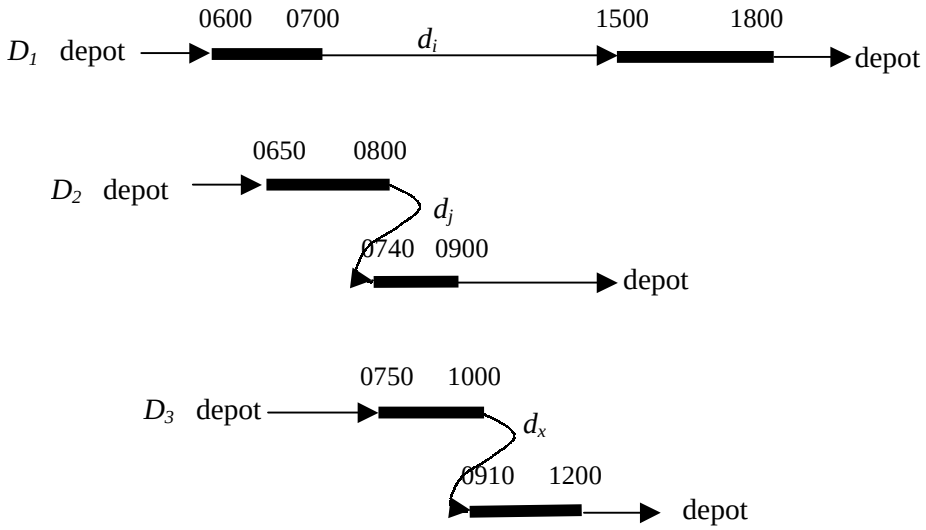


Figure 5.7: Illustration of three invalid links involved in *move*(*i*,*j*) and *move*(*i*,*x*)

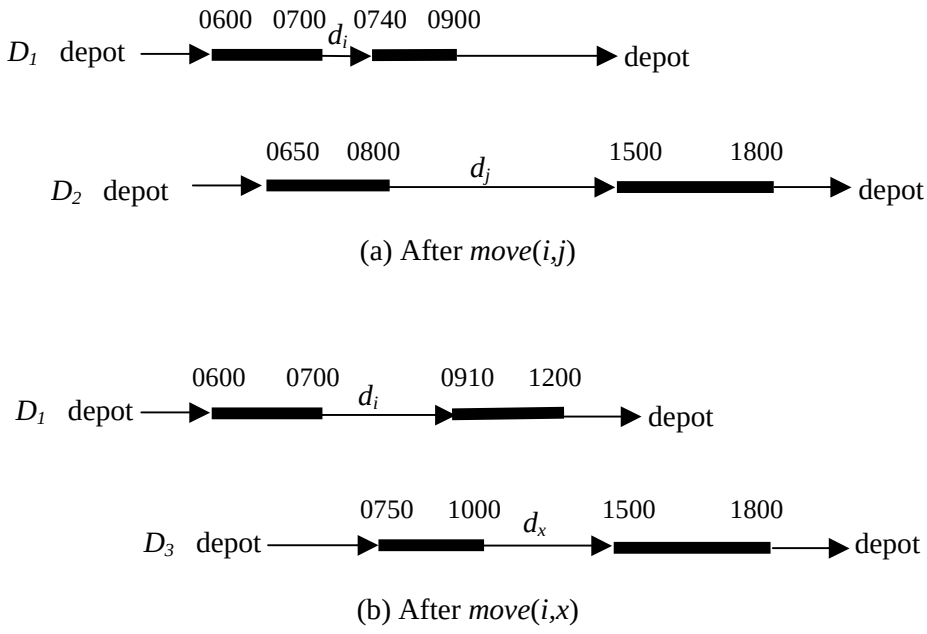


Figure 5.8: Illustration of the new links after two elite candidate moves

To avoid a very poor move being executed before there are still better moves in the candidate list, the elite candidate list approach is modified in the experiments as follows. After a move is executed and removed from the candidate list, the values of the remaining moves will be re-

computed, and then the moves are sorted by the order of move values. When the recent best most is below or equal to a given quality threshold, a new candidate list is constructed and the process repeats. In the experiments, the *quality threshold* was arbitrarily defined as where the move value is equal to a given small constant, say –500. If the constant is defined too high, all of the recorded moves in the candidate list may be below the quality threshold after a number of iterations. Then, the same elite candidate list will be constructed and the process repeats. This may form an endless loop if there is no limit on the number of iterations. If the constant is defined too low, e.g. –5000, the poor moves (which originally were elite moves but are turned into poor moves by the execution of their related moves) may be selected and executed at the very early stages.

This elite candidate list, comparing with the entire neighbourhood, reduces the number of moves evaluated at a given iteration. However, at each iteration, the recorded moves have to be re-evaluated and the quality of the best move cannot be ensured since some recorded moves may be influenced by some other previously executed moves, especially when a low quality threshold is defined. Experiments have shown that this method (in which the threshold constant is defined as –500) worked well at the first iterations, but became inefficient thereafter. The reason is that there would be more and more recorded moves under the quality threshold after a number of iterations. The reconstruction of an elite candidate list would become more frequent.

Several other candidate list approaches were designed as variants of the above candidate list approach, but none of them was satisfactory in terms of the quality of the best move and the computational effort. Consequently, we gave up the effort in designing a special candidate list (i.e. building a subset of the entire neighbourhood). Instead, HACS cuts down the computational effort by other means, which is described in the next section.

5.6.4.2 An efficient memory scheme for driver scheduling

Using the entire set of the neighbourhood as a candidate set (this can guarantee the quality of the best move at each iteration), HACS cuts down the computational effort by other means (diverting from using a subset of candidates), in which the number of solutions examined on a given iteration is greatly restricted as follows.

Each move defined in HACS only alters two duties; therefore, only two duties need to be re-evaluated after a move. The trial moves, not involving any link in the two altered duties, remain unaffected. The other trial moves, which are a minority, will be evaluated. Moreover, if the two links of a trial move are in the same duty, the move will not be considered and will be artificially assigned a very low move value. To fulfil this strategy, each link is assigned a *link-status*, which indicates whether a link is in a currently altered duty.

This memory scheme can guarantee the quality of the best move at an iteration while the computational effort is efficient. The implementation of the candidate list is discussed below. Different memory structures have been designed for the swapping move and the recutting move respectively.

- **Candidate list and tabu status for the swapping-link moves**

The candidate list is designed to be as large as the entire neighbourhood, which records all the move values in a lower triangular matrix $C = [c_{ij}]$, where $i>j$; $i = 2,3,\ldots,n$; $j = 1,2,\ldots,n-1$; and n is the number of links in the current schedule. As addressed above, it would be very time-consuming and unnecessary to update every value in C after a move. Only the moves

containing the recently visited links need to be costed again and *C* is updated accordingly. Each move in *C* has a corresponding tabu status, which indicates the number of iterations for which the move must remain tabu. If a move is not tabu, its tabu status is defined as zero. The tabu statuses and the move values could be recorded in the same matrix, such that the move values would be recorded in the lower triangle while the tabu statuses in the upper triangle accordingly. However, most of the moves would not be recorded as tabu at a given iteration. It would therefore be more efficient to record the tabu status of only the moves that are tabu in a tabu list.

The tabu list is designed as a fixed size (= tabu tenure) queue, which records *a* and *b* for a move *move*(*a*,*b*), and the tabu status of the move. The queue is initially empty. When the queue is full, its head record is removed and the tabu statuses of the remaining moves are decreased by one before a new record is added. The new added record always has its tabu status equal to the tabu tenure. If the tabu classification of a move is removed by an aspiration criterion, the corresponding record in the queue is transferred to the end of the queue, and the tabu status becomes equal to the tabu tenure. Any move not recorded in the queue is not tabu.

- **Candidate list and tabu status for the recutting-block moves**

Each link has two AROs (in the spell model) or two ROs (in the piece model), hence there are two possibilities to recut the blocks for each link. Suppose there are *n* links in a solution, the recutting-ARO operation on both AROs/ROs in each link can produce 2**n* potential schedules in the neighbourhood. The candidate list is designed to be as large as the entire neighbourhood, which records all the *links* and their corresponding *relevant links* (i.e. all potential moves), the *move values*, and the *tabu statuses* in a four-column matrix $C = [c_{ij}]$, where $i = 1,2,3,4$; $j = 1,2,\ldots,2n$.

The *tabu status* is initialised to 0 and is used to identify the ending iteration of the tabu tenure for the corresponding move. For a move, *iter* is used to denote the current iteration, at which this move is selected. The tabu status of this move is defined as:

$$tabu\ status(move) = iter + tabu\ tenure$$

In conjunction with the above definition, a move remains tabu if the tabu status of the move is not less than the current iteration.

5.7 Tabu search algorithms in HACS

TS algorithms are employed in the two phases of HACS: minimising penalty and minimising cost (see the general scheme of HACS presented in Section 4.5). The algorithms are the same in both phases but the objective functions are different accordingly. The pseudo codes of the algorithms are as follows, where S^* denotes the current best solution.

The algorithm for minimising penalty or cost

Given an initial solution *S* containing *n* duties, where *S* = (*P*, *Q*, *D*);

S^* := *S;*

Tabu-mechanism := *off;*

times := *0;*

While *times* < *Given-limit* do

    *A set of TS Algorithms** [1]

    If (The current solution *S* is better than the best solution *S**)

        *S** := *S;*

times := *0;*

else if (*Tabu-mechanism = off*)

Tabu-mechanism := *on;*

else

times := *times + 1;*

End if

2 End while

[*1] The set of TS algorithms varies according to the model, on which the HACS approach is based.

1. In the HACS approach based on the spell-model, the set contains the following algorithms:

   TS Algorithm – swapping-two-links (spell-links);

   TS Algorithm – swapping-two-spells;

   TS Algorithm – inserting-one-spell;

   TS Algorithm – recutting-block;

2. In the HACS approach based on the piece-model, the set contains the following algorithms:

   Tabu Search Algorithm – swapping-two-links (piece-links);

   Tabu Search Algorithm – swapping-two-pieces;

   Tabu Search Algorithm – inserting-one-piece;

   TS Algorithm – recutting-block (limited with WROs)*;*

3. In the HACS approach utilising the spell-based and piece-based operations together, the set contains the following algorithms:

   TS Algorithm – swapping-two-links (spell-links);

   TS Algorithm – swapping-two-spells;

   TS Algorithm – inserting-one-spell;

TS Algorithm – swapping-two-pieces;

TS Algorithm – inserting-one-piece;

TS Algorithm – recutting-block (limited within WROs)*;*

In the above minimising penalty or cost algorithm, multi-neighbourhoods (each operation decides a neighbourhood) are employed, each of which corresponds to a TS algorithm. Each TS algorithm is executed in turn until no further improvement can be made, taking the current best solution as its starting solution.

An alternative could be to use a compound move to coordinate the move operations. *A compound move* (Glover and Laguna, 1993) consists of a series of simpler components. For example, the rejection chain method used by Cavique et al (see Section 2.4.1) consists of three component operations: shift operation, cut operation and merge operation. Cavique's ejection chain method cannot compile duties consisting of more than two-spells. In driver scheduling, using a compound move would be very complicated if multi-spell duties are considered.

In the minimisation algorithm, a steepest descent method is first applied by switching off the tabu-mechanism. Since at the first stage of minimisation process the penalty or the cost could be easily reduced, the steepest descent method could be applied to reduce the penalty or the cost with less computational effort than the TS method. After the steepest descent method terminates at the local optimum, the TS algorithms are applied by switching on the tabu-mechanism.

The above minimisation algorithm contains a set of TS algorithms, each of which corresponds to a different move operation. They can be described by the following pseudo codes, where S is the current solution, S' is a trial solution, and S^* is the currently best solution.

Tabu Search Algorithm

Given an initial solution *S* containing *n* duties, where $S = (P, Q, D)$;

$S^* := S$;

$T := \emptyset$; (The tabu list is empty.)

iterator := *0;*

While *iterator* < *Given-iterator-limit* do

Search for the best move *move** such that *move** $\notin T$, or $f(S') < f(S^*)$;[*1]

$S := S'$;

If (*Tabu-mechanism* = *on*)

Update *T*; (put *move** into *T* and update the tabu status of moves *)*

End if

Update the set of links *D;*

Update the sets *P* and *Q*, if the current operation is recutting-block;

iterator := *iterator* + *1;*

If ($f(S) < f(S^*)$)

$S^* := S$;

End if

End while

[*1] In this tabu search algorithm, the *best move* is defined as the move with a highest move value amongst the moves either being non-tabu in the neighbourhood (*move** $\notin T$) or satisfying the *improved-best aspiration criterion* ($f(S') < f(S^*)$), i.e., a tabu classification on a trial move can be removed if the move yields a solution S' better than the best solution S^* obtained so far.

5.8 Conclusions

This chapter has presented the Tabu Search technique tailored for the driver scheduling problem. Multi-neighbourhood structures and an appropriate memory scheme, which are essential elements of Tabu Search, have been designed. An effective and efficient memory scheme (candidate list and tabu list) is specially designed for the driver scheduling problem. The multi-neighbourhood structures are proposed to diversify the search and enable a more comprehensive coverage of the search space. Experiments on the foregoing technique and some alternative designs are presented in the next chapter, comparing the results with best known solutions drawn from real-life data sets.

Chapter Six

Evaluation and Computational Experiments

6.1 Introduction

No clean proof of convergence for Tabu Search is known. Hertz et al (1997) describe Tabu Search to be like an engineering approach to large and difficult problems of optimisation, which may not possess the elegance of mathematics. Memory-based strategies are the hallmark of Tabu Search (Glover and Laguna, 1997). Memory, particularly in its adaptive forms, introduces too many degrees of freedom to be treated conveniently in 'theorem and proof' developments. Glover and Laguna (1997) suggest that researchers who prefer to restrict consideration to processes (and behaviour) that can be characterised by rigorous proofs must focus their efforts in other directions. Consequently, it is common to evaluate tabu search based approaches by

experiments. The quality of solutions and the computational time are the major concerns. Throughout this thesis, Relative Percentage Deviation (RPD) over best known solutions is employed to measure the quality of solutions.

$$RPD = (s-b)/b*100\% \quad (6.1)$$

where, s denotes the solution to be assessed while b denotes the best known solution.

The HACS approach presented in this thesis has been implemented using Borland C++ (Version 5.02) within an object-oriented framework. A large number of experiments have been carried out to assess HACS on the performance of individual components of the tabu search algorithm, the overall performance, and the capability of handling WROs. This chapter first presents the experiments related to the core of the tabu search technique in Sections 6.3 and 6.4. Experiments were first carried out on the spell-based operations to evaluate the neighbourhood structures and to compare tabu search with steepest descent search. Then, experiments were carried out on the piece-based operations and the combination of the spell-based and piece-based operations. The experiments related to WROs are presented in Sections 6.5 to 6.7. Finally the conclusions drawn from the experiments are provided at the end of this chapter.

6.2 Test problems and the best known solutions

The School of Computing, University of Leeds has a long history (more than thirty years) on transport scheduling. Their scheduling packages, e.g. BUSMAN (Chamberlain and Wren, 1992), OPENBUS (Wren and Kwan, 1999), and TRACS II, have been broadly installed in the UK and some other countries. Many practical problems have been successfully solved, from which we selected ten real-life driver scheduling problems as test data to evaluate the performance of the HACS system. The original problems have been accumulated over many years, during which the driver scheduling systems at Leeds have been evolving. We tested

HACS with the problems formatted for the recent TRACS II system. In the data set, the original A-WROs were replaced with one or two ROs each because TRACS II cannot handle A-WROs, while the original U-WROs were either kept (with a small gap) or removed (with a large gap) by splitting the blocks. The properties of the problems and the TRACS II solutions (i.e. the best known solutions to the problems) are displayed in Table 6.1. TRACS II is one of the most successful driver scheduling systems. It has been successfully installed for many bus companies and used by many train operating companies (see Kwan et al, 1999; Wren and Kwan, 1999). In tests against other established bus driver scheduling systems, TRACS II has always produced the most efficient schedules.

Table 6.1: Properties and best known solutions of test problems

Data Code	Data Name	Vehicle hours	No. ROs	No. Blocks	Best known solution			
					Potential Duties	No. Duties	Cost (hours)	Elapsed Time
D1	AG66	165:56	130	10	8090	24	180:18	3 mins
D2	AG62	178:46	200	13	58136	25	192:52	25 mins
D3	NURS	216:28	248	16	32904	31	235:05	10 mins
D4	GMB	227:52	175	21	11817	34	289:32	5 mins
D5	GOLD	294:10	289	23	27068	42	327:48	10 mins
D6	COM9	332:06	356	30	32000	39	363:26	10 mins
D7	TRMX	349:37	574	21	29500	49	408:47	10 mins
D8	LH7A	404:46	646	30	29021	54	427:04	10 mins
D9	LHM2	552.07	799	41	671541	80	632:32	Over 5 hours
D10	LHM1	786:47	859	63	999101	108	862:29	Over 15 hours

The elapsed times recorded in this thesis were actual clock times to obtain the solutions running on a 333 mHz personal computer, any exception will be indicated. The elapsed times for HACS were obtained by calling a built-in timing function in C-libraries. Since TRACS II does not report on elapsed times, and the duty generation and the ILP selection process were run separately, the elapsed times were estimated. In D9 and D10, slack parameters were used in the

TRACS II runs (on a Pentium III 1gHz personal computer), and much (over 85%) of the elapsed time reported was spent on generating the large number of potential duties.

In the test data, there are no A-WROs, which have been removed from the original raw data for the sake of TRACS II. We have to restore or add them to test the ability of HACS on handling WROs because tackling WROs is one of the most important objectives of this research. In the test data, the U-WROs with large gaps were not restored because they could not provide any benefit. The other U-WROs were restored by merging the split blocks. Some A-WROs were restored by checking the actual vehicle work. Unfortunately, most of the original A-WROs cannot be restored without the original raw data. We extended ROs into WROs artificially as follows. In practice, when a vehicle stops at a location, there is often a gap between the arrival time and the departure time, for instance 5 minutes or 8 minutes. These gaps usually vary in size. Without the benefit of a real data set, it might be more appropriate to extend ROs into WROs with variable durations, but in practice a fixed duration will still enable us to see the effect of altering the time of the ARO within the window (The HACS approach itself does not impose any restrictions on the durations of WROs). Therefore, we extended all the ROs in the testing problems into A-WROs with 5 minutes each, except those already expressed as WROs (their durations are variable) and the first and the last ROs in each block (their durations are zero). After this has been done, the problems D1 to D10 in table 6.1 become the problems with WROs and are labelled W1 to W10.

The experiments to be presented in this chapter were designed for evaluation of both HACS' ability to handle WROs and its tabu search technique. When evaluating the technique (see Sections 6.3 and 6.4), the problems D1 to D10 were used in order to compare the solutions with the best known solutions, in which no A-WROs were imposed. When evaluating the ability of HACS to handle WROs (see Sections 6.5 to 6.7), the problems W1 to W10 were employed.

6.3 HACS based on spell-model

To assess the performance of HACS, a number of experiments were devised in this section to:

- Compare the performance of different neighbourhood structures;
- Compare the tabu search technique and a steepest descent method;
- Evaluate the swapping processes.

Throughout this section, the experiments are based on the spell-model presented in Chapter 4.

6.3.1 Experiments on different neighbourhood structures

Given an objective function, different neighbourhood structures may give rise to different "landscapes" of local optima. In HACS, three spell -based swapping neighbourhood structures have been defined, each of which has different local optima and leads to a different solution. Table 6.2 compares the solutions obtained using each swapping neighbourhood structure separately while the recutting neighbourhood structure is always applied.

Obviously the quality of the solutions presented in Table 6.2 is poor. The average RPD in terms of number of duties is more than 6%. It is hard to conclude that one neighbourhood structure is superior to another.

The following experiments employed all three swapping operations and the recutting operation together, in which each neighbourhood structure is applied in turn, starting from the current best solution, until no further improvement can be achieved. The computational results presented in Table 6.3 show that HACS with all the swapping neighbourhood structures together produced better solutions than with only a single one. The application of multi-swapping neighbourhoods diversified the search and led to much better results. The average RPD of the HACS solutions

over the best known solutions was 3.24% in terms of number of duties and 1.27% in terms of cost. For the largest problem (D10) HACS produced a marginally better solution than TRACS II.

Table 6.2: Results obtained by HACS with different single swapping neighbourhood structure

Data	**Swapping-two-links**			**Swapping-two-spells**			**Inserting-one-spell**		
	No. Duties	Cost (hours)	Run Time (h:mm:ss)	No. Duties	Cost (hours)	Run Time (h:mm:ss)	No. Duties	Cost (hours)	Run Time (h:mm:ss)
D1	28	199.24	0:00:05	26	191.02	0:00:02	25	188.03	0:00:03
D2	26	192.37	0:00:14	26	199.21	0:00:08	26	200.41	0:00:13
D3	33	244.06	0:00:19	34	239.20	0:00:17	34	242.13	0:00:31
D4	36	285.57	0:00:10	36	289.16	0:00:10	37	292.37	0:00:17
D5	45	330.32	0:00:24	46	333.13	0:00:11	45	333.25	0:00:31
D6	44	388.13	0:00:16	44	384.00	0:00:23	46	393.12	0:00:36
D7	53	419.25	0:00:36	51	416.08	0:00:23	52	415.18	0:00:35
D8	58	457.10	0:01:11	58	457.22	0:00:59	59	470.33	0:01:11
D9	87	685.26	0:01:25	83	654.12	0:00:37	87	688.27	0:01:14
D10	113	889.11	0:03:32	110	865.26	0:02:15	118	927.02	0:05:10
RPD	8.19%			6.73%			8.52%		

Table 6.3: The solutions obtained by HACS with multi-neighbourhoods

Data	No. Duties	RPD (%)	Cost (hours)	RPD (%)	Run Time (h:mm:ss)
D1	25	4.17	182.26	1.18	0:00:07
D2	25	0.00	196.50	2.06	0:00:25
D3	33	6.45	239.33	1.90	0:00:30
D4	35	2.94	287.48	-0.60	0:00:19
D5	44	4.76	331.05	1.00	0:00:33
D6	42	7.69	371.15	2.15	0:00:54
D7	50	2.04	411.18	0.62	0:01:10
D8	55	1.85	438.52	2.76	0:01:43
D9	82	2.50	647.13	2.32	0:02:01
D10	108	0.00	856.11	-0.73	0:06:11
Avg.	3.24%		1.27%		

6.3.2 Experiments on a steepest descent method

As a more elaborate neighbourhood search method, tabu search can be described as a *steepest descent method* with a *dynamic neighbourhood* associated with an exploration history. Turning off the up-hill move operation (i.e. forbid any non-improved solutions), the tabu search approach becomes a steepest descent approach (i.e., from a given solution the move that leads to the best improved solution amongst the neighbourhood of the given solution is selected). The steepest descent method presented here was adapted from the HACS approach with all the neighbourhood structures. The computational results displayed in Table 6.4 show that the steepest descent method is inferior to the HACS approach based on the tabu search technique. However, the steepest descent method produced better solutions than the HACS approach with a single swapping neighbourhood structure.

Table 6.4: Results obtained by a Steepest Descent method

Data	No. Duties	RPD (%)	Cost (hours)	RPD (%)	Run Time (h:mm:ss)
D1	26	8.33	198.49	10.27	0:00:07
D2	27	8.00	192.15	-0.32	0:00:14
D3	33	6.45	244.28	3.99	0:00:25
D4	35	2.94	295.04	1.91	0:00:15
D5	45	7.14	334.31	2.05	0:00:26
D6	43	10.26	373.39	2.81	0:00:30
D7	51	4.08	415.46	1.71	0:00:30
D8	58	7.41	453.14	6.13	0:01:32
D9	83	3.75	650.27	2.83	0:01:53
D10	110	1.85	864.00	0.18	0:06:20
Avg.	6.02%		3.16%		

6.3.3 Evaluation of the swapping processes

In the tabu search algorithms of HACS, there are two main operations: *swapping* and *recutting*. Once a set of spells are fixed, the swapping function attempts to improve the solution as much as possible by re-linking the units of vehicle work. The recutting function attempts to improve the solution by re-selecting AROs. Both functions play very important roles in refining the schedule. Their performances will affect the solution quality of HACS although the cooperation of all components of HACS determines the ultimate solution quality. This subsection intends to investigate the individual performance of the tabu search swapping algorithms; the recutting function is not independently tested because there is no appropriate way to judge its individual performance.

To evaluate the tabu search swapping algorithms, we design an experiment in which the initial schedule formed is based on a 'good' set of spells. The swapping function, including all of the three swapping operations, is used solely to refine the schedule while the recutting-block and the adding-duties functions are switched off. The 'good' set of spells is taken from the best known solutions, i.e. TRACS II solutions.

- **Forming initial solutions based on the spells in TRACS II solutions**

The method of forming an initial schedule needs to be adjusted to use the spells in TRACS II solutions. In a TRACS II solution, some duties contain two-spells while the others may contain one, three or four spells (TRACS II only produces the duties with up to four-spells). However, HACS only initially forms the duties with up to two-spells. The number of duties in an initial schedule would be larger than that in the TRACS II solution containing three or four spell duties. Although it would be possible for the swapping process to reduce the number of duties

used, it is anticipated that the current HACS would have difficulty in reducing many duties. To focus on the performance of the tabu search swapping algorithms, the non-two-spell duties are adopted into the initial schedule while the spells in the other duties are re-coupled to form new two-spell duties. We use two different methods to couple the spells in order to test the swapping algorithms from different starting solutions. One method is the HACS' initialisation method. Another method forms an initial schedule by first sorting the spells by starting time, and then coupling two adjacent spells. The duties generated by one of the methods plus the original non-two-spell duties constitute an initial schedule. The HACS approach modified to test the swapping algorithms is denoted as HACS-1 or HACS-2 respectively according to the initialisation method incorporated.

- **Experiment 1**

The problem D6 was first selected for this experiment because the RPD, in terms of number of duties, over the best known solution is the highest amongst the test data listed in Table 6.3. It was expected that a better solution would be produced by using the 'good' set of spells in the best known solution. The best known solution to D6 costs 363 hours and 26 minutes and contains 39 duties, but only 18 of which are two-spell duties while the other duties are three-spell duties. Based on the 36 spells included in the 18 two-spell duties, two starting solutions and the corresponding final solutions are generated, which are summarised in Table 6.5, in which the similarity to the best known solution is analysed. In the initial solutions, the number of non-two-spell duties is presented in the first column, which is constant in both methods. When a solution is infeasible, the penalty of which is presented with 'p' following the number. The cost of a feasible solution is followed by a 'c'.

Table 6.5: The starting solutions and the corresponding final solutions to D6 obtained by HACS based on the spells in the TRACS II solutions.

System	Initial Schedule					Final schedule			
	No. Same duties		No. Diff. Duties	No. Invalid Duties	Total Cost or Penalty	No. Same duties	No. Diff. Duties	No. Invalid Duties	Total Cost or Penalty
HACS-1	21	13	5	0	364:18c	30	9	0	363:23c
HACS-2	21	0	18	18	13256p	18	21	3	147p

From Table 6.5, we can see that HACS-1 produces a solution 3 minutes cheaper than the best known solution starting from a more expensive feasible solution, and HACS-2 does not produce a feasible solution starting from a very poor initial solution. However, HACS-2 has improved the initial schedule considerably. In the initial schedule of HACS-2, every newly generated duty is illegal and different from the original duties. After swapping, most of the duties are still different. In HACS-1, the number of different duties has increased after swapping although the total cost is very close. The results indicate that there may be many possible combinations to form good feasible solutions based on the same set of spells.

- **Experiment 2**

In the previous experiments, the best known solution only contains 46% two-spell duties. This has reduced the difficulty to form a good initial schedule for HACS-1 because 54% of the duties are already formed and are valid. That case is not very common; in many cases two-spell duties are the majority in a solution. In this experiment, we select D7 as the test problem. The best solution to D7 has 408 hours and 47 minutes in cost and contains 49 duties, 48 of which are two-spell duties while only one is a three-spell duty. HACS-1 forms an initial schedule containing two infeasible duties. This initial schedule differs greatly from the best known solution. After swapping, a feasible solution is obtained, which is 48 minutes more expensive

and 55% of the duties are different from those in the best known solution. HACS-2 forms an extremely poor initial schedule, from which a considerably better solution is obtained after swapping although it is still infeasible. The details are shown in Table 6.6.

Table 6.6: The starting solutions and the corresponding final solutions to D7 obtained by HACS based on the spells in the TRACS II solutions.

System	Initial Schedule					Final schedule			
	No. Same duties		No. Diff. Duties	No. Invalid Duties	Total Cost or Penalty	No. Same duties	No. Diff. Duties	No. Invalid Duties	Total Cost or Penalty
HACS-1	1	12	36	2	935p	22	27	0	409:35c
HACS-2	1	0	48	48	29188p	22	27	3	223p

Similar to the previous experiment, this experiment shows that the HACS initialisation method produces a better initial solution than the method in HACS-2, and the better initial solution leads to a better final solution.

- **Experiment 3**

This experiment selects D9 as the test problem, which is the second largest amongst the test problems. The best known solution to D9 has 632 hours and 32 minutes in cost and contains 80 duties, 54 of which are two-spell duties. HACS-1 forms an initial schedule containing 8 infeasible duties. This initial schedule differs greatly from the best known solution. After swapping, a feasible solution is obtained, which is 1 hour and 56 minutes more expensive. HACS-2 forms an extremely poor initial schedule, from which a considerably better solution is obtained after swapping, which contains two invalid duties with 40 minutes penalty. The details are shown in Table 6.7. In this experiment, the solutions obtained by HACS-1 and HACS-2 are considerably different from the TRACS II solution, and the number of different duties is not explicitly counted.

Table 6.7: The starting solutions and the corresponding final solutions to D9 obtained by HACS based on the spells in the TRACS II solutions.

System	Initial Schedule					Final schedule	
	No. Same duties		No. Diff. Duties	No. Invalid Duties	Total Cost or Penalty	No. Invalid Duties	Total Cost or Penalty
HACS-1	26	15	39	8	8420p	0	634:28c
HACS-2	26	0	54	54	51546p	2	40p

The foregoing experiments show that the HACS-1 solutions to the three problems are of a quality comparable to the best known solutions. Hence, the tabu search swapping algorithms have reached an acceptable level of performance because we do not expect an optimal solution. As for the HACS-2 solutions, which are not good enough, the tabu search swapping algorithms do improve the extremely poor starting solutions considerably. For local search methods, better initial solutions usually lead to better final solutions.

6.3.4 Summary

This section has presented a series of experiments on the HACS approach with the spell-model. The experiments have shown the importance of multi-neighbourhood structures and the tabu search framework. The multi-neighbourhood structures used in combination are superior to a single neighbourhood structure used in isolation, and the tabu search technique is superior to the steepest descent method. Compared with TRACS II, HACS can produce solutions considerably quicker and is very easy to manipulate because many parameters in TRACS II requiring careful setting are not needed. The experiments on solely evaluating the swapping function have shown that the swapping function has reached an acceptable level of performance. Given a good set of spells, HACS could generate a good solution. However,

without this condition, the HACS solutions may be unsatisfactory. The next section will focus on the enhancement of the recutting function by incorporating the piece-model.

6.4 HACS with combined utilisation of the spell model and the piece model

It was anticipated that the piece-based operations would improve the results because more ROs would be considered than the spell-based recutting operation, in which only the ROs adjacent to the current AROs are considered as potential AROs. A number of experiments have been done, in which the same memory structure was used since links are still the basis of the memory structure.

As discussed in Section 5.5, an enhanced HACS approach, having a combined utilisation of the spell-model and the piece model, is anticipated to allow all ROs a chance of being used while the driver work would not become too fragmented. In this enhanced approach, the five swapping operations (*swapping-two-links*, *swapping-two-spells*, *inserting-one-spell*, *swapping-two-pieces*, *inserting-one-piece*) and a *recutting-block* (only applicable to WROs) operation are applied in turn. The test problems in this set of experiments do not contain WROs, hence no blocks need to be re-cut. The computational results (presented in Table 6.8) show:

- The HACS approach with the spell-based and piece-based operations combined produced better solutions than the other variant HACS approaches.
- In most cases, the improved HACS solutions to the test problems are close to the best known solutions. The average RPD over the best known solutions is 1.36% in terms of number of duties and 1.30% in terms of total cost.

- For the largest problem in the table, HACS produced a better solution than TRACS II.
- The HACS approach can produce solutions very quickly.

Table 6.8: Results obtained by the **HACS** approach with the spell-based and piece-based operations combined

Data	No. Duties	RPD (%)	Cost (hours)	RPD (%)	Run Time (h:mm:ss)
D1	24	0	182:29	1.21	0:00:24
D2	25	0	196:14	1.74	0:00:54
D3	32	3.22	239:15	1.77	0:01:02
D4	34	0	291:56	0.83	0:00:36
D5	44	4.76	330:18	0.76	0:00:59
D6	40	2.56	362:48	-0.17	0:01:40
D7	49	0	414:05	1.30	0:02:53
D8	56	3.70	450:34	5.50	0:09:17
D9	81	1.25	643:41	1.76	0:03:33
D10	106	-1.85	847:55	-1.69	0:17:54
Avg. RPD	**1.36**		**1.30**		

So far, the HACS approach, incorporating the spell-based and piece-based operations, has been developed. This approach is superior to the other variant HACS approaches and has produced the solutions close to the best known solutions. It will be applied in the following experiments in order to test its ability in handling WROs.

6.5 Demonstration of handling WROs

This section will demonstrate by experiments that HACS can take advantage of WROs. A small artificial test problem was first created for the demonstration. In order to assess the quality of

the solution, two methods were used to apply TRACS II to obtain solutions to the test problem, although TRACS II cannot handle WROs practically.

6.5.1 The TEST problem and the HACS solution

The artificial problem created is called TEST, in which the set of labour agreement rules was selected from a real-life problem while the vehicle work was fictitious and contained fourteen pieces of work distributed in three blocks as shown in Figure 6.1. In the labour agreement rules, three types of duty: *split duty*, *straight duty* with mealbreak and *non-mealbreak duty*, are allowed, and the maximum length of spreadover is 12:30, 8:06, or 5:30 respectively. For each duty, the maximum length of spell is stipulated as 5 hours.

Figure 6.1 also illustrates the solution obtained by HACS. The schedule, produced within less than one minute, contains three duties and costs 24 hours and 13 minutes.

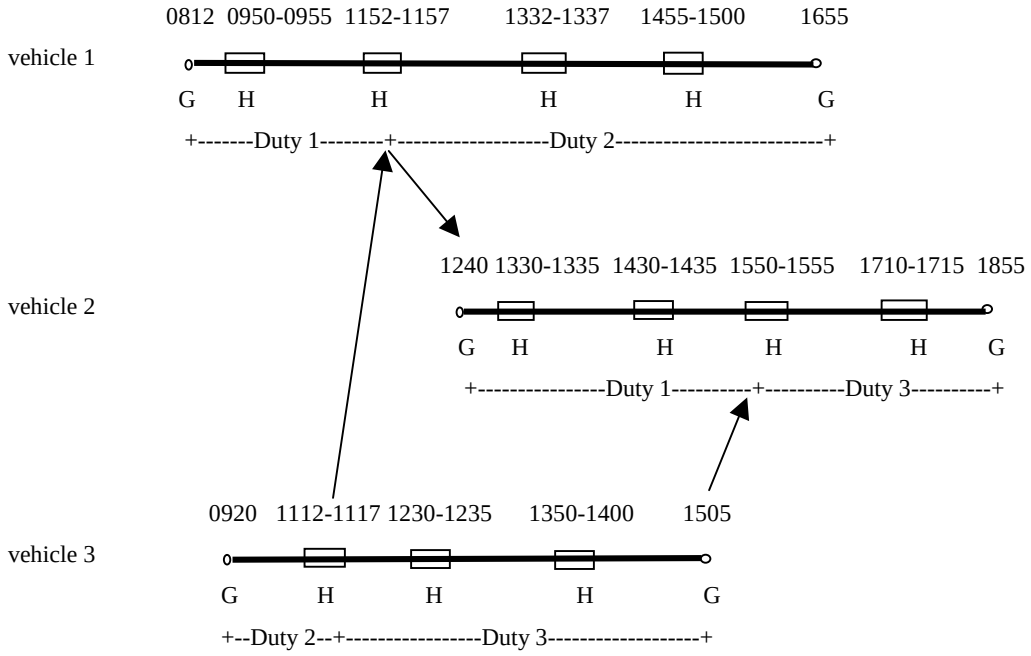


Figure 6.1: The TEST problem and the HACS solution

6.5.2 TRACS II solutions to the TEST problem and analyses

It has already been mentioned that none of the existing systems can directly handle windows of relief opportunities. There are only two possible ways for TRACS II to tackle the TEST problem. They are discussed separately below.

1) Shrink a WRO into a RO by selecting only the arrival time as a relief time

This is a common method for the existing approaches to solve practical driver scheduling problems. After the change, the blocks included in the TEST problem were converted into those as displayed in Figure 6.2, where no WROs existed.

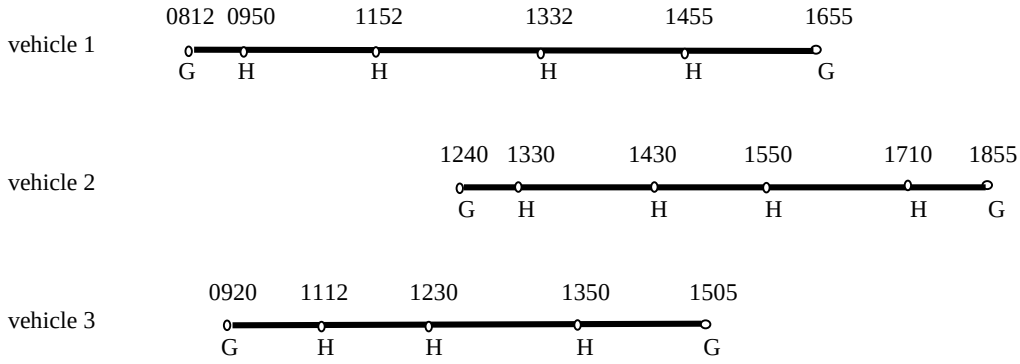


Figure 6.2: Blocks in the TEST problem after shrinking each WRO into a RO

TRACS II was then run and produced a solution as shown in Figure 6.3. This solution contained four duties with a total wage cost of 24 hours and 58 minutes, which was worse than the HACS solution in the terms of both the number of duties and the cost.

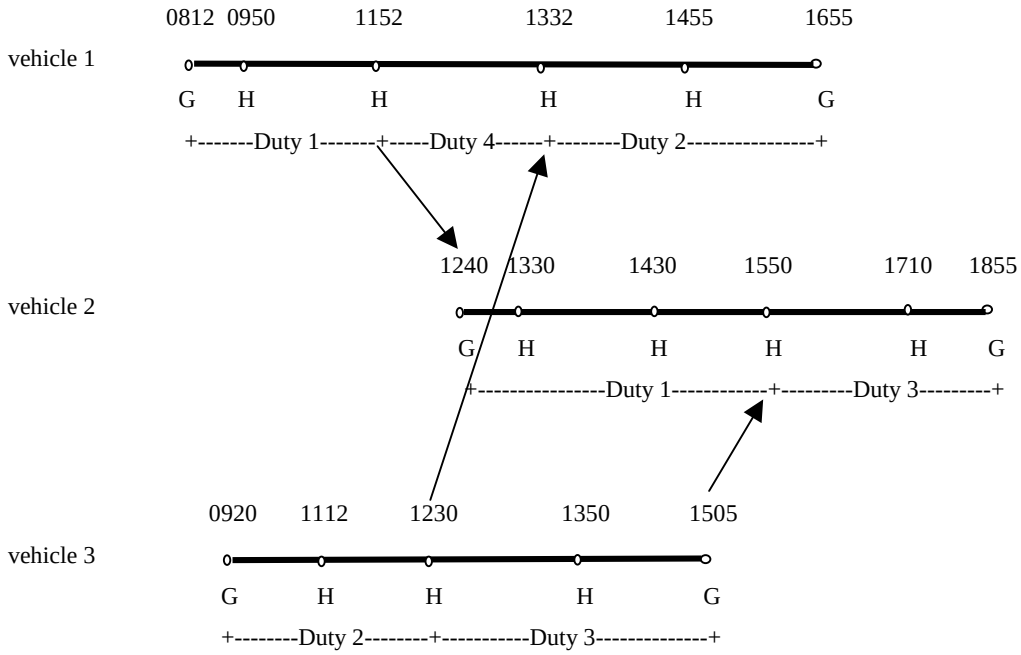


Figure 6.3: TRACS II solution to the TEST problem after shrinking each WRO into a RO

Comparing the TRACS II solution with the HACS solution, we found that the TRACS II solution contained seven spells while the HACS solution had six. Further investigation found that TRACS II partitioned the first block into three spells while HACS divided it into two spells. The explanation is: the length of the first block is *8:43*, which is longer than the maximum length of spell (five hours). The block therefore has to be partitioned into at least two spells. The longest early spell could be from 8:12 to 13:12 (the sum of 8:12 plus 5:00). However, since 13:12 is not a relief time, an earlier relief time 11:52 was selected. Similarly, the longest spell covering the end of the block could be from 11:55 to 16:55. Since 11:55 is not a relief time, the later time 13:32 was selected. The work between 11:52 and 13:32 was not covered; hence, three spells were essential to cover all the work in the first block.

However, in the original TEST problem presented in Figure 6.1, the time 11:55 in the first block was a relief time falling in the time window from 11:52 to 11:57. It was possible to partition the block into two at the time 11:55, 11:56, or 11:57. The WROs enabled the use of one fewer spell and to produce the better solution. This experiment has exemplified the improvement that WROs may bring and that shrinking WROs might lose some opportunities to produce a better solution.

2) Expand explicitly each WRO into a sequence of individual ROs at one-minute intervals

This method would not normally be practical because it will increase considerably the number of ROs, and is thus not suitable for the generate-and-select based approaches, e.g. TRACS II, in which the magnitude of a problem is largely dependent on the number of ROs. Since the TEST problem is artificially tiny in terms of vehicle work to be covered, it would be worth trying. Two experiments were carried out: one used slack filtering rule parameters while another used tight parameters.

Theoretically, using slack filtering parameters enlarges the search space providing more chance of producing a good solution. HACS does not need any filtering rules because it only refines the small solution set of duties constructed. However, TRACS II has to apply filtering rules so that the set of potential duties generated would not be prohibitively large. In the duty generation process, TRACS II first generates the set of potential legal *spells*, the length of each is limited in a time range stipulated by parameters. It then generates the set of potential legal *stretches* based on the set of spells. Finally by combining the stretches a large set of potential duties is generated. There are upper bound limits for the number of potential spells, stretches generated.

To run TRACS II in a similar condition to HACS, slack filtering rule parameters were used in the first experiment. Unfortunately, TRACS II terminated before finishing the duty generation process because the number of stretches exceeded TRACS II's limit (899 spells and 26637 stretches had been formed before the termination).

The second experiment was then designed using tighter parameters in order to reduce the number of stretches to be formed. The minimum lengths of spells and stretches were set to one hour and three hours respectively. A solution was obtained in about ten minutes, which contained four duties and the total cost in hours and minutes was 23:53. Figure 6.4 demonstrates the schedule, where the times underlined are active relief times. *Duty 1* is a straight duty with a meal break and the other duties are non-mealbreak duties, amongst which *duty 4* contains two spells.

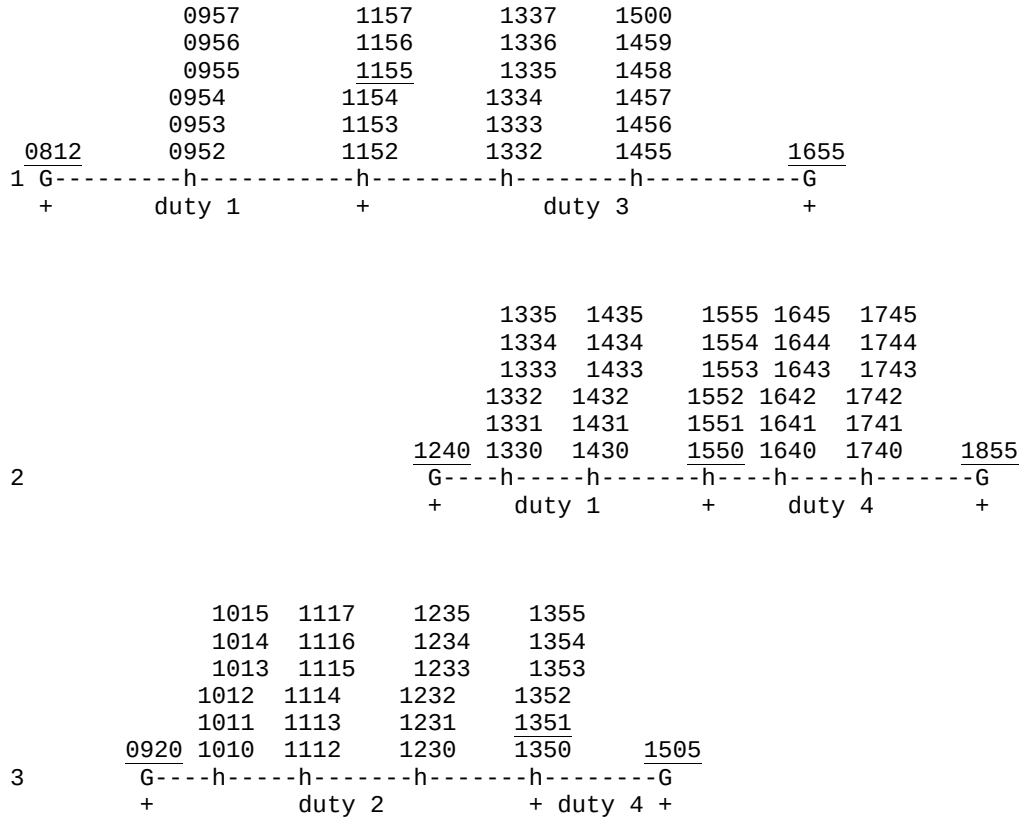


Figure 6.4: TRACS II solution to the TEST problem after expanding each time window into individual minutes

In the foregoing experiment, WROs were represented in the same way as in HACS, but the solution contained one more duty. The reason was investigated below.

Comparing the solutions, we found in the HACS solution that the shortest stretch, which contained only one spell, was 1 hour and 55 minutes. This stretch was shorter than the minimum length of stretch stipulated in TRACS II. Therefore, it was impossible for TRACS II to produce such a solution.

6.5.3 Summary

This section has demonstrated that HACS can take advantage of WROs, but the generate-and-select approaches, e.g. TRACS II, cannot because shrinking WROs or using tight parameters compromises the quality of solutions while the slack parameters cause the set of potential stretches or duties generated to be too large even for trivial problem instances with WROs. It is impractical to expand explicitly each time window into individual minutes using the existing generate-and-select approaches.

6.6 Experiments on real-life problems with WROs

Further experiments were carried out on the real-life problems presented in Table 6.1. Since these problems (D1 to D10) do not contain WROs, HACS was run on the problems W1 to W10, which were derived from D1 to D10 by imposing WROs (see Section 6.2). This enables us to see further the ability of HACS to tackle WROs. The computational results are displayed in Table 6.9.

Table 6.9: HACS solutions involving WROs

DATA	HACS solutions (to the problems W1 to W10)					Best solutions to D1 to D10 by TRACS II (see Table 6.1)		Best solutions to D1 to D10 by HACS (see Table 6.8)	
	No. Duties	RPD (%)	Cost (hours)	RDP (%)	Elapsed Time	No. Duties	Cost (hours)	No. Duties	Cost (hours)
W1	24	0	181:50	0.85	0:00:08	24	180:18	24	182:29
W2	25	0	193:56	0.55	0:00:37	25	192:52	25	196:14
W3	32	3.23	240:42	2.39	0:01:03	31	235:05	32	239:15
W4	34	0	288:31	-0.35	0:00:19	34	289:32	34	291:56
W5	44	4.76	330:31	0.83	0:00:39	42	327:48	44	330:18
W6	39	0	364:25	0.27	0:00:46	39	363:26	40	362:48
W7	49	0	410:08	0.33	0:04:05	49	408:47	49	414:05
W8	54	0	433:29	1.50	0:07:48	54	427:04	56	450:34
W9	81	1.25	645:04	1.98	0:08:09	80	632:32	81	643:41
W10	104	-1.89	830:18	-2.08	0:31:16	108	862:29	106*	847:55
Avg. RPD	0.74%		0.63%						

* The best known solution to D10 was produced by HACS. The best known solutions to the other problems (D1 to D9) were generated by TRACS II.

The computational results presented in table 6.9 show:

- HACS produced better solutions to all the ten problem instances with WROs imposed (W1 to W10) than to those without A-WROs (D1 to D10). The best HACS solutions to D6, D8 and D10 were considerably improved, in which the number of duties was reduced, by taking advantage of WROs.
- In most cases, the HACS solutions to the test problems are close to the best known solutions. The average RPD over the best known solutions is 0.74% in terms of number of duties and 0.63% in terms of total cost.
- For the largest problem (W10) in table 6.9, HACS produced a better solution than TRACS II. (A larger problem usually has more WROs, which provide more chances of improving the schedule).

6.7 Experiments to improve best known solutions

Theoretically, WROs provide more chances of compiling a good schedule. This experiment investigates if HACS could improve the best known solutions by taking advantage of WROs. This experiment was carried out using the best known solutions as initial solutions for HACS. The best known solutions were obtained by solving the test problems without WROs. Table 6.10 shows the results, from which it can be seen that the best known solutions to all the test problems have been improved by HACS, taking advantage of WROs, in terms of cost. The average RPD of the HACS solutions over the best known solutions is –0.85%. This implies that HACS could be used as a post process for TRACS II or other systems to improve their solutions by WROs. One of the most recent experiments, which was on scheduling train staff for London Underground Metropolitan Line, provides further proof.

Table 6.10: HACS solutions based on the best known solutions

Data	HACS solutions				Best known solutions	
	No. Duties	Cost (hours)	RDP (%)	Elapsed Time (h:mm:ss)	No. Duties	Cost (hours)
W1	24	179:08	-0.65	0:00:03	24	180:18
W2	25	190:25	-1.27	0:00:05	25	192:52
W3	31	234:41	-0.17	0:00:05	31	235:05
W4	34	279:41	-3.40	0:00:08	34	289:32
W5	42	326:57	-0.26	0:00:07	42	327:48
W6	39	361:23	-0.56	0:00:16	39	363:26
W7	49	407:36	-0.29	0:00:20	49	408:47
W8	54	424:54	-0.51	0:00:32	54	427:04
W9	80	625:16	-1.15	0:00:52	80	632:32
W10	106	845:44	-0.26	0:01:41	106	847:55
Avg. RPD	0%	–0.85%				

The exercise on scheduling train staff for London Underground Metropolitan Line was mainly carried out by Professor A. Wren and his colleagues. Since there were some time windows involved, it was anticipated that HACS could improve the best solution generated by TRACS II.

This problem had 569 ROs and 17 relief points, of which two were depots. Amongst the ROs, 37 were WROs, which together contained 4 hours and 45 minutes of daily work in one vehicle. Figure 6.5 illustrates the vehicle work diagram, where the dotted lines denote WROs. This vehicle operates a shuttle service from CLFX to another point and back. Most of the round journeys last 22 minutes each and then wait at CLFX for 8 minutes.

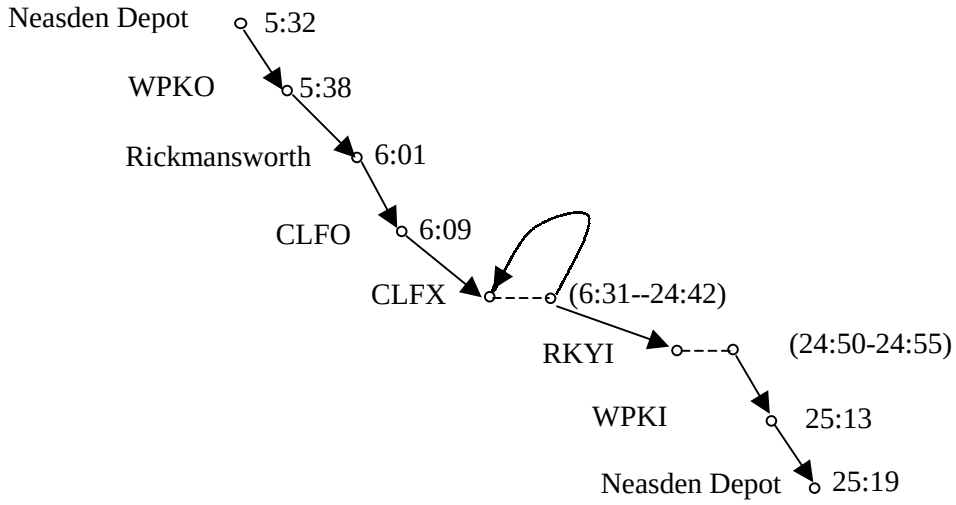


Figure 6.5: Illustration of a vehicle work with WROs in the London Underground Metropolitan Line

There were some special conditions: the ability to sign on or sign off at relief points other than depots at certain times of day, and the ability to provide different latest sign off times at certain relief points.

To solve the problem with the special conditions, an intelligent manipulation of the data for TRACS II was under taken while each WRO was shrunk into two ROs. The best solution obtained by TRACS II at the time used 79 duties with a cost of 598 hours and 17 minutes. It was anticipated that the TRACS II solution might be improved in cost by taking advantage of WROs using HACS. Starting from the TRACS II solution, HACS generated a solution with 596 hours and 10 minutes in cost. The HACS solution improved the TRACS II solution by saving 2 hours and 7 minutes in terms of cost while the number of duties used was the same.

6.8 Conclusions

This chapter has presented a series of experiments to evaluate the HACS approach. The tabu search technique in HACS has been evaluated first. The experiments have shown the importance of the multi-neighbourhood structures and the tabu search framework. By combining spell-based and piece-based operations, HACS has improved the chance of all ROs to be used while the driver work would not become too fragmented and the computation is efficient. The computational results are generally nearly as good as the TRACS II solutions.

After testing the technical aspects of HACS, this chapter has also presented the experiments on handling WROs. The demonstration on solving the TEST problem shows that HACS can take advantage of WROs but TRACS II cannot. We can deduce that it is impractical to expand explicitly each time window into individual minutes using the existing generate-and-select approaches.

The further experiments on solving the ten real-life problem instances with WROs have shown that HACS is able to handle WROs while producing solutions generally close to the TRACS II

solutions using significantly shorter computational time. By employing the best known solutions as initial solutions, HACS can improve the best known solutions by taking advantage of WROs. The case of London Underground Metropolitan Line indicates that HACS has practical significance for improving existing solutions by taking advantage of WROs.

From all the experiments presented in this chapter, we can see that HACS can produce solutions in one-go and is considerably quicker than TRACS II, an ILP-based generate-and-select approach. Theoretically, using slack labour rule parameters enlarges the search space providing more chance to produce a good solution. However, the duty generation process could take a very long time when the labour rule parameters are slack. Hence, TRACS II uses some parameterised filtering rules to prevent too many potential duties from being generated. The HACS approach does not need any filtering rules because it only refines the small solution set of duties constructed. Although the current HACS solutions are generally only close to the TRACS II solutions, HACS has the potential of being developed to yield solutions better than TRACS II because its search space is not limited by the filtering rules and it can take advantage of WROs. This feature (no soft rules under consideration) also provides HACS another advantage: easy to manipulate.

Chapter Seven

Conclusions

7.1 Summary

The bus and train driver scheduling problem has been presented. While reviewing the approaches for solving the driver scheduling problem, the shortcomings of the current approaches have been identified. Early heuristic approaches are heavily dependent on problem domain knowledge and hard to adapt between organisations. Although generate-and-select approaches, e.g. TRACS II and HASTUS, are at present the most successful for driver scheduling, they still cannot be regarded as black boxes that produce working schedules and they cannot guarantee to yield optimal solutions within practical computational limits. Windows of relief opportunities, which often exist in real-world driver scheduling problems, cannot be handled by them. Filtering rules are often applied so that the set of potential duties generated would not be prohibitively large. The adjustment of the parameterised filtering rules is sometimes frustrating and time-consuming. The ILP-based approaches cannot ensure a

solution at the first attempt, and it is likely that there will always be problems for which decomposition is necessary on size grounds.

Constructive approaches have been investigated in this thesis, under which filtering rules are unnecessary. Initial research was aimed at investigating the applicability of a 2-opt heuristic local search method for driver scheduling. The 2-opt heuristic approach is much less dependent on domain knowledge than the early heuristic approaches and is easier to adapt to different situations. The initial research was limited to form the duties with up to two-spells. The 2-opt approach can produce solutions very quickly and is very easy to manipulate because there is no filtering rule parameters to be adjusted by users. Development and experiments on the 2-opt approach have demonstrated the feasibility of a constructive heuristic approach for driver scheduling and inspired the tabu search approach.

Benefiting from the experience in the 2-opt heuristics, two models have been built for the driver scheduling problem with WROs. In these models the labour agreement rules governing the legality and payment of duties have been formulated by quantified penalty and cost functions. These functions can be easily adjusted to different situations. A schedule is modelled as a set of links, which constitutes the basis for the refining operations. Adjusting the set of links may improve the schedule, and any ROs in windows of relief opportunities may have the chance to be selected as AROs during the scheduling process. Hence, it is possible for the constructive approach to handle WROs based on the models. In addition, there is no limit on the number of spells in a duty as necessitated in the TRACS II system.

Based on the models, a constructive approach, HACS, employing Tabu Search, has been developed. In the HACS approach, the tabu search technique has been appropriately tailored for

the driver scheduling problem with WROs. A large number of experiments have been undertaken, of which the computational results have been presented in detail.

7.2 Research Contributions

This research has investigated two local search methods, *2-opt heuristics* and *Tabu Search*, for the bus and train driver scheduling problem with WROs. The problem without WROs is a special case, in which all WROs have zero durations. Despite extensive literature search and discussions with colleagues, it appears that this is the first time that research is focussed on time windows in driver scheduling.

Two models for bus and train driver scheduling problem have been built. Based on the models a heuristic driver scheduling approach, called HACS, has been developed. The core of the HACS approach is the Tabu Search technique, which has been appropriately tailored for the driver scheduling problem with WROs. HACS does not need any filtering rules to reduce problem size. Consequently, HACS is easy to manipulate and has the potential of being developed to yield schedules better than TRACS II, although the current HACS solutions are only close to, generally not as good as, the TRACS II solutions.

Experiments have shown that the HACS tabu search approach has met the proposed goals:

1. HACS can take advantage of WROs, which cannot be practically handled by other existing systems;
2. HACS does not use any artificial rules and is easy to manipulate (unlike generate-and-select approaches, which have to apply a set of filtering rules to reduce problem size,

and require manipulation of these parameterised filtering rules; this is often frustrating to human schedulers).

In addition, HACS has the following features:

a) HACS can compile duties, in which the number of spells is unlimited. (Generate-and-select approaches limit the number of spells in a duty so that the set of potential duties generated would not be prohibitively large);
b) HACS can produce solutions quickly (considerably quicker than TRACS II).
c) It can be applicable to different transport operators since the most common labour agreement rules have been considered in formulating the penalty and cost functions.
d) During the scheduling process, HACS produces a sequence of intermediate solutions. Some infeasible intermediate solutions might be useful for the scheduler who might be able to adjust the operations to reduce the number of drivers.

Experiments on the ten sets of real-life problems have shown that HACS has produced results closely comparable to the best known solutions. The average RPD over the best known solutions is 1.36% in terms of total number of duties and 1.30% in terms of total cost. The results would be better if the WROs could be restored (this has been demonstrated by experiments). With better starting solutions, e.g. the blocks are first cut according to the TRACS II solutions, the results could be improved further. It should be noted that HACS has only been developed after about three person-years work while perhaps 100 person-years have been devoted to the developments of TRACS II (Kwan et al, 2000).

It has practical significance that HACS can improve the TRACS II solutions by taking advantage of WROs within a short computational time. Consequently, HACS could be

incorporated into existing systems such as TRACS II to improve the solution by taking advantage of WROs. The extra relief opportunities, within time windows, chosen for use in the improved solution could be inserted into the original TRACS II data for re-running. The whole process could be repeated until HACS cannot make any further improvement.

The research presented in this thesis is concerned with bus and train driver scheduling; however, it is anticipated that the modelling method, the idea of allowing infeasible intermediate solutions, and the tailored tabu search technique (especially the efficient memory scheme and multi-neighbourhood structures) could be applied to many combinatorial problems, such as vehicle routing problems, job shop scheduling and travelling salesman problems.

7.3 Future Work

HACS could benefit from further development in both methods of constructing an initial schedule and adding duties. HACS might be advanced by more complex compound moves, involving intermixing the swapping moves and the recutting moves.

In a large operating area, an individual driver may not know all the routes. However, the usual assumption is that each depot, and therefore its drivers, is associated with knowledge of a set of routes. In a vehicle schedule, specific driver route knowledge may be required for individual pieces of work. Similarly in the train situation, each block in a train schedule may have specific traction knowledge required of the drivers. Further research may cater for route knowledge, traction knowledge and other practical driver scheduling constraints, so that HACS would be more versatile for the transport industry.

In real-time operation, vehicle delay, accident or some other unexpected events may happen. This may need the adjustment of the vehicle schedules and the driver schedules. Further research may adapt HACS to produce quickly a revised driver schedule with least changes according to the newly adjusted vehicle schedule.

Combining *tabu search* with other meta-heuristics such as *simulated annealing* and *genetic algorithms* is also worthy of further research.

Bibliography:

Aarts, E. and Lenstra, J. K. (eds.) (1997). *Local Search in Combinatorial Optimization*. Wiley, Chichester.

Aarts, E., Korst, J.H.M. and van Laarhoven, P.J.M. (1997). Simulated annealing. In: Aarts, E. and Lenstra, J. K. (eds.) *Local Search in Combinatorial Optimization*, Wiley, Chichester, pp. 91-120.

Ball, L. O., Bodin, L. D. and Greenberg, J. (1985). Enhancements to the RUCUS 2 crew scheduling system. In: Rousseau, J.-M. (ed.) *Computer Scheduling of Public Transport 2*, North-Holland, pp. 279-293.

Blais, J.-Y. and Rousseau, J.-M. (1988). Overview of HASTUS Current and Future Versions. In: Daduna, J.R. and Wren, A. (eds.) *Computer-Aided Transit Scheduling*, Springer Verlag, pp.175-187.

Bodin and Bergman (eds.) (1975). *Preprints of the International Workshop on Automated Techniques for Scheduling of Vehicle Operators for Urban Public Transportation Services*, Chicago.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. In: Glover, F., Laguna, M., Taillard, E., and de Werra, D. (eds.) *Tabu Search. Annals of operations research*, vol. 41, J.C. Baltzer AG, Science Publishers, Basel. pp, 157-183.

Carraresi, P. and Rousseau, J.-M. (1982). Relaxation Approaches to Large Scale Bus Driver Scheduling Problems, *Transportation Research*, Vol. 16B, pp. 383-397.

Cavique, L., Rego, C. and Themido, I. (1999). Subgraph ejection chains and tabu search for the crew scheduling problem, *Journal of the Operational Research Society*, Vol. 50, pp. 608-616.

Chamberlain, M. and Wren, A. (1992). Developments and recent experience with the BUSMAN and BUSMAN II systems. In: Desrochers, M. and Rousseau, J.-M. (eds.) *Computer-Aided Transit Scheduling*, Springer Verlag, pp. 1-15.

Curtis, S. D. (2000). *Constraint satisfaction approaches to bus driver scheduling*. PhD thesis, School of Computing, University of Leeds, UK.

Curtis, S.D., Smith, B.M. and Wren, A. (1999). Forming bus driver scheduling using constraint programming. In: *Proceedings of the first international conference on Practical Application of Constraint technologies and Logic Programming PACLP99*, pp. 239-254.

Curtis, S.D., Smith, B.M. and Wren, A. (2000). Constructing driver schedules using iterative repair. In: *Proceedings of the second international conference on Practical Application of Constraint technologies and Logic Programming PACLP2000*, pp. 59-78.

Daduna, J.R., Branco, I. and Paixao, J.M.P. (eds.) (1995). *Computer-Aided Transit Scheduling, Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport*, Springer-Verlag.

Daduna, J.R. and Wren, A. (eds.) (1988). *Computer-Aided Transit Scheduling, Proceedings of the Fourth International Workshop on Computer-Aided Scheduling of Public Transport*, Springer-Verlag.

Desrochers, M., Gilbert, J., Sauve, M. and Soumis, F. (1992). CREW-OPT: subproblem modelling in a column generation approach to urban crew scheduling. In: Desrochers, M. and Rousseau, J.-M. (eds.) *Computer-Aided Transit Scheduling*, Springer Verlag, pp. 395-406.

Desrochers, M. and Rousseau, J.-M. (eds.) (1992). *Computer-Aided Transit Scheduling, Proceedings of the Fifth International Workshop on Computer-Aided Scheduling of Public Transport*, Springer-Verlag.

Desrochers, M. and Soumis, F. (1988). CREW-OPT: crew scheduling by column generation. In: Daduna, J.R. and Wren, A. (eds.) *Computer-Aided Transit Scheduling*, Springer Verlag, pp. 83-90.

Desrochers, M. and Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem, *Transportation Science*, Vol. 23, pp. 1-13.

Dias, T. G., Ferreira, J. V. and Cunha, J. F. (2000). Evaluating a DSS for operational planning in public transport systems: ten years of experience with the GIST system. Presented in: *the eight International Conference on Computer-Aided Scheduling of Public Transport*, 21st-23rd June, 2000, Berlin.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on System, Man, and Cybernetics*, Vol. 26(1), pp. 29-41.

Elias, S.E.G. (1964). The use of digital computers in the economic scheduling for both man and machine in public transportation. *Special report no. 49*, Kansas State University Bulletin.

Elias, S.E.G. (1966). A mathematical model for optimising the assignment of man and machine in public transit "run-cutting". *Research Bulletin no. 81*, West Virginia University Engineering Experiment Station.

Faigle, U. and Kern, W. (1992). Some convergence results for probabilistic tabu search. *ORSA Journal on Computing*, Vol. 4, pp. 32-37.

Fores, S. (1996). *Column generation approaches to bus driver scheduling*. PhD thesis, School of Computing, University of Leeds, UK.

Fores, S. and Proll, L. (1998). Driver scheduling by integer linear programming - the TRACS II approach. In: Borne, P., Ksouri, M. and El Kamel, A. (eds.) *Proceedings CESA' 98 Computational Engineering in Systems Applications, Volume 3 Symposium on Industrial and Manufacturing Systems*, pp.213-218.

Fores, S., Proll, L. and Wren, A. (1998). A column generation approach to bus driver scheduling. In: Bell, M. H. G. (ed) *Transportation Networks: Recent Methodological Advances*, Pergamon, pp.195-208.

Fores, S., Proll, L. and Wren, A. (1999). An Improved ILP System for Driver Scheduling. In: N. H. M. Wilson (ed.) *Computer-Aided Transit Scheduling*, Springer Verlag, pp.43-62.

Fores, S., Proll, L. and Wren, A. (2001). Experiences with a flexible driver scheduler. To appear in: Voß, S. and Daduna, J.R. (eds.) *Computer-Aided Scheduling of Public Transport*, Springer Verlag.

Forsyth, P. and Wren, A. (1997). An ant system for driver scheduling. *Technical Report 97.25*, School of Computing, University of Leeds, UK.

Gendreau, M., Laporte and Potvin, J.-Y. (1997). Vehicle routing: modern heuristics. In: Aarts, E. and Lenstra, J. K. (eds.) *Local Search in Combinatorial Optimization*, Wiley, Chichester, pp. 311-336.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, Vol. 13, pp. 533-549.

Glover, F. (1989). Tabu search - part 1. *ORSA Journal on Computing*, Vol. 1, pp.190-206.

Glover, F. (1990). Tabu search - part 2. *ORSA Journal on Computing*, Vol. 2, pp. 4-32.

Glover, F. (1996). Ejection chains, reference structures and alternating path methods for travelling salesman problems. *Discrete Applied Mathematics*, Vol. 65, pp. 223-253.

Glover, F. and Laguna, M. (1993). Tabu Search. In: Reeves, C.R. (ed.), *Modern heuristic techniques for combinatorial problems*, Blackwell Scientific Publications, pp.70-150.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

Glover, F., Laguna, M., Taillard, E., and de Werra, D. (eds.) (1993). *Tabu Search*. In: Hammer, P. L. (editor-in-chief) *Annals of operations research*, vol. 41, J.C. Baltzer AG, Science Publishers, Basel, Switzerland.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. Presented at*: the Congress on Numerical Methods in Combinatorial Optimisation*, Capri.

Hertz, A. and de Werra, D. (1991). The tabu search metaheuristic: how we used it. *Annals of Mathematics and Artificial Intelligence*, Vol. 1, pp. 111-121.

Hertz, A., Taillard, E. and de Werra, D. (1997). Tabu search. In: Aarts, E. and Lenstra, J. K. (eds.) *Local Search in Combinatorial Optimization*, John Wiley and Sons Ltd., Chichester, England, pp. 121-136.

Hildyard, P. M. and Wallis, H. V. (1981). Advances in computer assisted runcutting in North America. In: Wren, A. (ed.) *Computer Scheduling of Public Transport*, North-Holland, pp.183-192.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, University of Michigan Press.

Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, Vol. 49, pp. 291-307.

Kirkpatrick, S., Gelatt, C.D. Jr and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, Vol. 220, No. 4598, pp. 671-680.

Kling, R. M. and Banerjee, P. (1987). ESP: A new standard cell placement package using simulated evolution. In: *proceedings of the 24th ACM/IEEE design automation conference*, Miami Beach, FL, pp. 60-66.

Kwan, A.S.K. (1999). *Train Driver Scheduling*. PhD thesis, School of Computing, University of Leeds, UK.

Kwan, A.S.K., Kwan, R.S.K., Parker, M.E. and Wren, A. (1999). Producing train driver schedules under differing operating strategies. In: Wilson, N.H.M. (ed.), *Computer-Aided Transit Scheduling*, Springer-Verlag, pp.129-154.

Kwan, A.S.K., Kwan, R.S.K. and Wren, A. (1999a). Driver scheduling using genetic algorithms with embedded combinatorial traits. In: Wilson, N.H.M. (ed.) *Computer-Aided Transit Scheduling*, Springer Verlag, pp. 81-102.

Kwan, R.S.K. and Rahin, M.A. (1999). Object oriented bus vehicle scheduling - the BOOST system. In: Wilson, N.H.M. (ed.), *Computer-Aided Transit Scheduling*, Springer-Verlag, pp. 179-194.

Kwan, R.S.K., Wren, A. and Kwan, A.S.K. (2000). Hybrid genetic algorithms for scheduling bus and train drivers. In: *Proceedings of the 2000 congress on evolutionary computation*, vol. 1, pp. 285-292. La Jolla, CA, USA.

Layfield, C. J., Smith, B. M. and Wren, A. (1999). Bus relief opportunity selection using constraint programming. In: *Proceedings of practical application of constraint technologies and logic programming conference - PACLP99*, The Practical Application Company Ltd, pp. 537-552.

Lessard, R., Rousseau, J.-M. and Dupuis, D. (1981). HASTUS I: A Mathematical Programming Approach to the Bus Driver Scheduling Problem. In: Wren, A. (ed.) *Computer Scheduling of Public Transport*, North-Holland, pp.255-267.

Li, J. and Kwan, R.S.K. (2000). A genetic algorithm with fuzzy comprehensive evaluation for driver scheduling. *Technical Report 2000.17,* School of Computing, University of Leeds, UK.

Li, J. and Kwan, R.S.K. (2001). A fuzzy simulated evaluation algorithm for the driver scheduling problem. To appear in: *proceedings of the 2001 IEEE Congress on Evolutionary Computation*, May 27-30, Seoul, Korea.

Lin, S. (1965). Computer solutions of the travelling salesman problem. *Bell System Technical Journal*, Vol. 44, pp. 2245-2269.

Lin, S. and Kernighan, B.W. (1973). An effective heuristic algorithm for the travelling salesman problem. *Operations Research,* Vol.21, pp.498-516.

Lourenço, H. R., Paixão, J. P. and Portugal, R. (2000). Multiobjective metaheuristics for the bus-driver scheduling problem. Presented in: *the eight International Conference on Computer-Aided Scheduling of Public Transport,* 21st-23rd June, 2000, Berlin.

Luedtke, L. K. (1985). RUCUS II: A Review of System Capabilities. In: Rousseau, J.-M. (ed.) *Computer Scheduling of Public Transport 2*, North-Holland, pp. 61-116.

Madureira, A. M. (1999). Meta-heuristics for the single-machine scheduling total weighted tardiness problem. In: *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, Porto, Portugal, pp. 405-410.

Manington, B. and Wren, A. (1975). A general computer method for bus crew scheduling. In: Bodin and Bergman (eds.) *Preprints of the International Workshop on Automated Techniques for Scheduling of Vehicle Operators for Urban Public Transportation Services,* Chicago.

Mori, H and Matsuzaki, O. (2001). Embedding the priority list into tabu search for unit commitment. In: *Power Engineering Society Winter Meeting, 2001 IEEE*, Vol. 3, pp. 1067-1072.

Muhlenbein, H. (1997). Genetic algorithms. In: Aarts, E. and Lenstra, J. K. (eds.) *Local Search in Combinatorial Optimization*, John Wiley and Sons Ltd., Chichester, pp. 137-172.

Nara, K., Hayashi, Y., Ikeda, K. and Ashizawa, T. (2001). Application of tabu search to optimal placement of distributed generators. In: *Power Engineering Society Winter Meeting, 2001 IEEE*, Vol. 2, pp. 918-923.

Okada, M., Jaji, K. and Fukushima, M. (1998). Probabilistic analysis of 2-opt for travelling salesman problems, *International Journal of Systems Science*, Vol.29, No, 3, pp. 297-310.

Ozdemir, H.T. and Mohan, C. K. (2000). Evolving schedule graphs for the vehicle routing problem with time windows. In: *Proceedings of the 2000 congress on evolutionary computation*, vol. 2, pp. 888-895. Piscataway, NJ, USA.

Parker, M.E. and Smith, B.M. (1981). Two Approaches to Computer Crew Scheduling. In: Wren, A. (ed.), *Computer Scheduling of Public Transport*, North-Holland, pp. 193-221.

Parker, M.E. and Wren, A. and Kwan, R. S. K. (1995). Modelling the scheduling of train drivers. In: Daduna, J. R., Branco, I. and Paixao, J.M.P. (eds.) *Computer-Aided Transit Scheduling,* Springer Verlag, pp. 359-370.

Pham, D.T. and Karaboga, D. (2000). *Intelligent optimisation techniques*, Springer-Verlag.

Rayward-Smith, V.J., Osman, I.H., Reeves, C.R. and Smith, G.D. (eds.) (1996). *Modern Heuristic Search Methods*, John Wiley and Sons Ltd., Chichester, England.

Reeves, C.R. (ed.) (1993). *Modern heuristic techniques for combinatorial problems*, Blackwell Scientific Publications.

Reeves, C.R. (ed.) (1996). Modern Heuristic Techniques. In: Rayward-Smith, V.J., Osman, I.H., Reeves, C.R. and Smith, G.D. (eds.) *Modern Heuristic Search Methods*, John Wiley and Sons Ltd., Chichester, England.

Reinelt, G. (1994). *The traveling salesman computational solutions for TSP applications. Lecture Notes in Computer Science, 840*. Springer-Verlag.

Rousseau, J.-M. (ed.) (1985). *Computer Scheduling of Public Transport 2, Proceedings of the Third International Workshop on Computer-Aided Scheduling of Public Transport*, North-Holland.

Rousseau, J.-M. and Blais, J.-Y. (ed.) (1985). HASTUS: An Interactive System for Buses and Crew Scheduling. In: Rousseau, J.-M. (ed.) *Computer Scheduling of Public Transport 2*, North-Holland, pp. 45-60.

Rousseau, J.-M. and Desrochers, M. (1995). Results obtained with CREW-Opt, a column generation method for transit crew scheduling. In: Daduna, J. R., Branco, I. and Paixao, J.M.P. (eds.) *Computer-Aided Transit Scheduling*, Springer Verlag, pp. 349-358.

Rousseau, J.-M., Lessard, R. and Blais, J.-Y. (ed.) (1985). Enhancements to the HASTUS Crew Scheduling Algorithm. In: Rousseau, J.-M. (ed.) *Computer Scheduling of Public Transport 2*, North-Holland, pp. 295-310.

Schmidt, J.W. and Knight, R.L. (1981). The status of computer-aided scheduling in North America. In: Wren, A. (ed.) *Computer Scheduling of Public Transport*, North-Holland, pp. 17-22.

Sherali, H. D. (1982). Equivalent weights for lexicographic multi-objective programs: characterizations and computations. *European Journal of Operations Research*, Vol. 18, pp. 57-61.

Shen, Y. and Kwan, R.S.K. (2000). A Tabu Search algorithm with 2-opt moves for bus and rail driver scheduling. Presented in: *the Eleventh Young Operational Research Conference*, 28th-30th March, Fitzwilliam College, Cambridge, UK.

Shen, Y. and Kwan, R.S.K. (2000a). Tabu Search for time windowed public transport driver scheduling. *Technical Report 2000.14*, School of Computing, University of Leeds.

Shen, Y. and Kwan, R.S.K. (2001). Tabu Search for Driver Scheduling. To appear in: Voß, S. and Daduna, J.R. (eds.) *Computer-Aided Scheduling of Public Transport*, Springer Verlag.

Shen, Y. and Kwan, R.S.K. (2001a). A constructive approach to bus and train driver scheduling. In: *Proceedings of the IIE Annual Conference 2001*, May 20-23, 2001, Dallas, Texas.

Smith, B.M. (1986). *Bus Crew Scheduling Using Mathematical Programming*. PhD thesis, School of Computing, University of Leeds, UK.

Smith, B.M. (1988). IMPACS – A Bus Crew Scheduling System Using Integer Programming. *Mathematical Programming*, Vol. 42, pp. 181-187.

Smith, B.M. and Wren, A. (1981). VAMPIRES AND TASC: Two Successfully Applied Bus Scheduling Programs. In: Wren, A. (ed.), *Computer Scheduling of Public Transport*, North-Holland, pp. 97-124.

Smith, B.M. and Wren, A. (1988). A bus crew scheduling system using set covering formulation. *Transportation Research*, Vol. 22A, pp. 97-108.

Tian, Y., Sannomiya, N. and Xu, Y. (2000). A tabu search with a new neighbourhood search technique applied to flow shop scheduling problems. In: Proceedings of the 39th IEEE conference on Decision and Control, Sydney, Australia, pp. 4606-4611.

Verhoeven, M.G.A., Aarts, E.H.L. and Swinkels, P.C.J. (1995). A parallel 2-opt algorithm for the Travelling Salesman Problem, *Future Generation Computer Systems*, Vol.11, pp. 175-182.

Voß, S. and Daduna, J.R. (eds.) (2001). *Computer-Aided Scheduling of Public Transport, Proceedings of the Eighth International Conference on Computer-Aided Scheduling of Public Transport*, Springer-Verlag (in press).

Weaver, A (1968). *A crew scheduling program*. In: Carr, J.D. (ed.), *Proceedings of a PTRC seminar*, Birkbeck College London, pp.48-50.

Weaver, A (1972). *Bus crew scheduling*. MPhil thesis, School of Computing, University of Leeds, UK.

Weaver, A and Wren, A. (1970). Bus and crew scheduling. *Report in Operational Research Unit*, School of Computing, University of Leeds, UK.

Weaver, A and Wren, A. (1972). Bus crew scheduling by computers. *Report in Operational Research Unit*, School of Computing, University of Leeds, UK.

De Werra, D. and Hertz, A. (1989). Tabu search techniques: a tutorial and an application to neural networks. *OR Spektrum*, Vol. 11, pp. 131-141.

Wilhelm, E. B. (1975). Overview of the RUCUS package driver run cutting program – RUNS. Presented at: *the Workshop on Automated Techniques for Scheduling of Vehicle Operators for Urban Public Transportation Services* , Chicago, US.

Willers, W. P. (1995). *Improved algorithms for bus crew scheduling*. PhD thesis, School of Computing, University of Leeds, UK.

Wilson, N.H.M. (ed.) (1999). *Computer-Aided Transit Scheduling, Proceedings of the Seventh International Workshop on Computer-Aided Scheduling of Public Transport*, Springer-Verlag.

Wren, A (1968). *A review of computer scheduling of buses and crews*. In: Carr, J.D. (ed.), *Proceedings of a PTRC seminar*, Birkbeck College London, pp. 42-47.

Wren, A. (1972). Bus Scheduling: An Interactive Computer Method. *Transportation Planning and Technology*, Vol. 1, pp. 115-122.

Wren, A. (ed.) (1981). *Computer Scheduling of Public Transport, Proceedings of the Second International Workshop on Computer-Aided Scheduling of Public Transport*, North-Holland.

Wren, A. (1998). *OR6 - scheduling and constraint management, Lecture notes*, School of Computing, University of Leeds, UK.

Wren, A. and Kwan, R.S.K. (1999), Installing an Urban Transport Scheduling System. *Journal of Scheduling*, vol.2, pp. 3-17.

Wren, A. and Rousseau, J.-M. (1995). Bus Driver Scheduling – An overview. In: Daduna, J.R., Branco, I. and Paixao, J.M.P. (eds.) *Computer-Aided Transit Scheduling*, Springer Verlag, pp. 173-187.

Wren, A. and Smith, B.M. (1988). Experiences with a Crew Scheduling System Based on Set Covering. In: Daduna, J.R. and Wren, A. (eds.) *Computer-Aided Transit Scheduling*, Springer Verlag, pp. 104-118.

Wren, A. and Smith, B.M. and Miller, A. J. (1985). Complementary approaches to crew scheduling. In: Rousseau, J.-M. (ed.) *Computer Scheduling of Public Transport 2*, North-Holland, pp. 262-278.

Wren, A. and Wren, D. O. (1995). A genetic algorithm for public transport scheduling. *Computers and Operations Research*, Vol. 22, pp. 101-110.

Glossary

In public transport scheduling areas, different transport companies may have different meanings for some terms. Below is a glossary of some transport scheduling terms used in this thesis.

Active relief opportunity (ARO): A special **relief opportunity**, in which a driver is actually relieved.

Attended window of relief opportunities (A-WRO): A **window of relief opportunities**, during which a vehicle must be attended by a driver.

Block: A sequence of journeys to be operated by one vehicle during one day, beginning with a pull-out from, and ending with a pull-in to a **depot**.

Depot: A place where vehicles and drivers are dispatched from at the start of their work period and returned to at the end of their daily work.

Duty: The work to be performed by a driver during one day from signing on until signing off at a **depot**.

Join up (join-up): The time allowance between two **spells** of work for the driver to get off from the current vehicle to take over the next vehicle, which is not a **mealbreak**.

Labour agreement rules: Rules agreed by government, the transport operators and the union, which govern the legality of duties.

Meal break (mealbreak): The time allowance between two **spells** of work for the driver to travel to the canteen having a meal and then travel to take over the next vehicle.

Over cover (over-cover): When **a piece of work** is assigned to two or more driver **duties**.

Piece of work (piece): An indivisible period of driving work, between two **windows of relief opportunities.**

Relief opportunity (RO): A time/location pair, at which a driver can be relieved.

Relief point: Designated locations on vehicle work where drivers may be relieved.

Relief time: A time when a vehicle passes a **relief point**.

Spell: a section of continuous vehicle work to be operated by a driver without a break.

Spilt duty: a type of duty that has a longer spreadover than standard duties and has a long break in the middle.

Stretch: The time period that a driver can work without a mealbreak, consisting of one or more successive spells in a duty.

Time window: The time range between the arrival time and the departure time when a vehicle stops at a location.

Unattended window of relief opportunities (U-WRO): A **window of relief opportunities**, during which the vehicle is left without a driver.

Window of relief opportunities (WRO): The time range that a vehicle remains at a **relief point**, during which the driver can be relieved.